Proc. Indian Acad. Sci. (Engg. Sci.) Vol. 3, Pt. 2, July 1980, pp. 89–128. © Printed in India.

Fault-tolerant computing concepts for aerospace applications—a survey

A PEDAR and V V S SARMA*
Systems Engineering Division, National Aeronautical Laboratory, Bangalore 560 017
*School of Automation, Indian Institute of Science, Bangalore 560 012

MS received 19 September 1979; revised 4 February 1980

Abstract. A fault-tolerant computing system performs its intended functions irrespective of the occurrence of certain failures. As the system becomes more and more reliable with built-in fault-tolerance, its analysis, validation and comparison with another system become formidable tasks. Such a study involves issues such as fault classification, figures of merit, fault-tolerant architectures, coverage estimation and automated methods of reliability evaluation. This paper provides a comprehensive survey of all these topics. The new concept of performability, which combines both the performance and the reliability of a system, and the configuration optimisation of a gracefully degradable computing system are also discussed.

Keywords. Fault-tolerant computing; fault-tolerance; fault analysis; redundant digital structures; coverage; reliability modelling; configurational optimisation; aerospace application.

1. Introduction

In recent years, the computer has become an essential equipment in critical manned/unmanned space missions, critical biomedical equipment and also in life-critical control systems such as those used in control configured vehicles (CCV) and active control aircraft (ACA). In all the above applications, it is apparent that a small computing error and/or the failure of the computer could result in the loss of life, expensive equipment or the output of years of research. The need to minimise the risk associated with computer failure and to provide reliable and uninterrupted computing, in spite of one or several failures in computing systems, has led to the evolution of a new class of computing systems known as fault-tolerant computing (FTC) systems.

Flight vehicle profitability and flight safety are the two criteria which ultimately dictate the design of a flight vehicle computer system (Bjurman *et al* 1976). The computer performance requirements should normally reflect the goals formulated for the flight vehicle's operational level. The operational level goal requirements such as despatch critical functions, flight mandated functions and pilot workload relief functions, are brought down to the requirements of flight crucial functions, flight critical functions and noncritical functions respectively at the computer level. From the flight safety point of view, the failure rate should be less than 1×10^{-9} for crucial functions and within 1×10^{-9} to 1×10^{-5} for critical functions. Airborne computer systems should have channel self-dependence, channel integrity, negligible

A list of symbols appears at the end of the paper.

fault propagation, voting at output, auto start/restart and a recovery time ranging from less than few milliseconds to a maximum of 1 s depending on the application. For space-borne computers most of the above considerations apply. In addition, the fact that no repair is possible and that the computing power requirement at the end of the mission is almost equal to that at the start allows no degradation. It is suggested that self-checking computing module (SCCM) architecture is better suited for space borne applications (Rennels 1978).

Fault-tolerant computing has been defined as 'the ability to execute specified algorithms correctly regardless of hardware failures and/or software failures' (Avizienis 1977). In general, failure of a computing system can be characterised as the deviation from its intended behaviour. When a computing system is characterised by a set of sequential states, the failure can be attributed to the movement of the system from the normal correct states to those states which cannot exist for that particular operation (Meyer 1976). However, the class of 'gracefully degradable computing systems' need a better definition which takes into account its ability to perform correctly, in spite of failures, in a degraded mode. Therefore, depending on the user requirement and use environment, the degree of fault-tolerance varies.

Fault avoidance and fault-tolerance are the two approaches used in the design of reliable computing systems. The fault avoidance approach relies on the use of nearly perfect components in the system design which can never be achieved in practice (Avizienis 1978). In this approach, it is obvious that the failure is inevitable and that the time of occurrence of the event is only postponed. The fault avoidance approach always has to be backed by manual maintenance, which may not always be possible for reasons such as cost, delay and inaccessibility. Fault tolerant approach, on the other hand presumes that the system is unreliable and tries to provide reliable computing, in spite of failures, with some form of protective redundancy. Fault-tolerance and fault avoidance are basic and complementary approaches and the best designed computing system exploits the advantages of both the methods with the resources allocated between them in an optimal manner.

This paper provides a state-of-the art survey of various aspects of fault-tolerant computing. § 2 deals with measures that are used to describe FTC systems. The fault classification and the mathematical modelling of a system with faults are dealt with in § 3. Different configurations and methods of building in fault tolerance are covered in § 4. § 5 gives a detailed analysis of coverage, the most important factor in FTC system design. Computer automated and unified reliability modelling approaches are surveyed in § 6. The new parameter 'performability' which aptly describes a gracefully degradable computing system, and its evaluation are given in § 7. The method of optimising the number in each kind of resource available in the multiprocessor system by taking the interaction of resources into account is given in § 8. Fault-tolerant software techniques are reviewed in § 9 and our concluding remarks in § 10.

2. Figures of merit for fault-tolerance

There are various parameters by which the effectiveness of fault-tolerance of FTC systems can be quantified. The first measure is the number of tolerable faults before the system goes down. A fail operational-fail operational-fail operational (FO-FO-FO)

fault-tolerant system means that the system can tolerate three failures before the system completely fails (Perry *et al* 1972). This parameter thus gives the user a feel of the extent of redundancy employed. The user is normally interested in knowing the mission success probability over a period of time. The usual parameters used for this purpose are mean time between failures (MTBF), reliability and survivability. Normally, reliability considers only the permanent hardware faults. The parameter survivability, however, takes into account both permanent and transient faults in quantifying the fault tolerance. It has been shown (Bouricius *et al* 1969) that the measure MTBF is misleading when it is used for comparing two highly reliable fault-tolerant computing systems because the mission time of interest is very much smaller than the time span over which the MTBF is evaluated. System reliability $R(t)$ is yet another parameter. For highly reliable FTC systems, failure probability rather than the reliability itself is better suited for comparison purposes. Another parameter is the length of time T, during which the reliability level is more than that required by the mission, (i.e.) T for which $R(t) \geqslant R_{\text{specified}}$ over $0 \leqslant t \leqslant T$. The ratio of the above times for any two systems, known as the mission time improvement factor (I) is a good parameter for comparison.

A new concept in fault tolerant computing is that of gracefully degradable computing systems in which the system can exhibit one of several worthwhile levels of performance (as viewed by the user) which depends on the history of the structure of the computers, use environment during the specified 'utilisation period' and the user requirements (Borgerson & Freitas 1975; Jacques Losq 1977). These computing systems need entirely new parameters for their evaluation and comparison as we can find that performance evaluations (of a fault-free system) will generally not suffice since structural changes, due to faults, may be the cause of degraded performance. By the same token, traditional views of reliability no longer suffice since ' success ' can take on various meanings and in particular it need not be identified with ' absence of system failure '. Meyer (1978) has introduced a unified performance reliability measure referred to as ' performability ' which relates directly to the system effectiveness and is a proper generalisation of both performance and reliability. A critical step in performability modelling is the introduction of the concept of a 'capability function' which relates low level system behaviour to user-oriented levels of performance. A detailed discussion of this evaluation is given in § 5.

3. Fault classification and analysis

The fault set of any FTC system falls into either physical faults (failures due to hardware/software and interference) or man-made faults (failures due to imperfect specification, design production and man-machine interaction) (Avizienis 1978). Physical faults, in turn, are classified according to the form of hazard causing them (random, induced or specification), duration (transient or permanent), extent (level of occurrence, their nature—local or distributed), their value (determinate or indeterminate) and finally their frequency of occurrence.

Transient faults are caused by temporary failure or interference in use environment and are characterised by the duration, frequency and the maximum duration possible (Holick 1963). The permanent faults are due to random hazards whose

effect is permanent. The error propagation in any system is as follows. Physical fault or interference causes error signal which produces erroneous data. This false data cause anomalous module behaviour that results in system malfunction.

A mathematical model of a digital computing system with faults is given by Meyer (1976). In this model the basic computer is defined as a discrete time, finite state sequential machine, whose state at any time is described by a transition function. In mathematical form the computer is described by

$$C = (X, Q, \Delta),$$

where X is a finite, nonempty set, the input set of C; Q is a finite, nonempty set, the state set of C and Δ is a sequence of functions

$$\Delta = (\delta_0, \delta_1, \delta_2 \ldots .),$$

where $\delta = Q \times X \to Q$ the transition function of C at time $i (i \in T)$; (i.e.) $\delta_i(q, a)$ is the state of the system at $(i+1)$th time interval ($q \in Q$ is the state of C at time i, $a \in X$ is the input at time i).

The physical failure can be transient, permanent or a combination of both. The occurrence of failure changes the transition function. The failure occurring in the time interval i and $i+1$ can be interpreted in a more meaningful way as follows, by the triplet $f = (\beta, \pi, i)$ where β is the transition function that the failing system exhibits while the failure takes place and π is the transition function that the system exhibits after the failure has taken place (i.e.) the transition function δ_j with faults is described by

$$\delta_j^f = \begin{cases} j & 0 \leqslant j < i \\ \beta & j = i \\ \pi & j > i. \end{cases}$$

So the computer with faults is denoted by

$$C^f = (X, Q, \Delta^f) \text{ where } \Delta^f = (\delta_0^f, \delta_1^f, \ldots).$$

Thus the computer with faults is a triplet (C, F, ϕ) where $C \in \mathscr{C}$ and C is time invariant. F is a set of faults of C, and F contains at least one null fault and $\phi : F \to C$ where $\phi(f) = C^f$. In the formulation of system success, the users' success criteria and the probabilistic nature of both faults and environment must be taken into account. The probabilistic nature of the use environment is more aptly described by the probability space (E, ξ, P_E); similarly the probabilitistic nature of the faults of the computer C is represented by another probability space $(F, \mathscr{F}, P_f)$.

When the nature of faults and use environment are specified, they can be combined into a single product space given by (G, g, P) where

$$G = E \times F,$$
$$g = \{G' \mid G' \in G\},$$
$$P = g \to [0, 1].$$

P is the unique probability measure that satisfies the condition $P(\{(e, f)\}) = P_E\{e\} \cdot P_F\{f\}$ for all $(e, f) \in G$. User success criteria is another parameter set that has to be taken into account in considering the system with faults. A fault which occurs at time t may or may not affect the system. Conventional analysis, however, treats the system as having failed irrespective of the effect of the fault on the system. Meyer has introduced the σ tolerance relation concept to take care of such situations.

The final computation-based reliability expression that takes into account all the above factors is given as

$$R_\sigma(C) = P(H),$$

where $H = \{(q, i, x, f) \mid$ the computation $(q, i, x, \alpha_q^t(1, x)$ is a success$\}$.

The present authors have also given nonhomogeneous Markov model (figure 1) for an ROM (read only memory) which is used for storing noninterruptible type of real time software (Pedar & Sarma 1978). The state sets that describe the above model are all good states, nonaffecting failed states and failed states. This nonhomogeneous Markov model is used to evaluate the computation-based reliability and also the computation-based mean time to failure (MTTF) which are given by the following expressions

$$R_i^C(T) = \exp\,[-\lambda T\{(i+1)K - i(i+1)/2\}],$$

and

$$\mathrm{MTTF}_i^C = R_{i-1}^C(T)/\lambda(K-i),$$

where K is the number of cells in ROM and i is the time instant.

4. Fault-tolerant digital computing system architecture

The fault-tolerance approach applied in any of the existing fault-tolerant computing systems normally falls into any one or a combination of component redundancy,

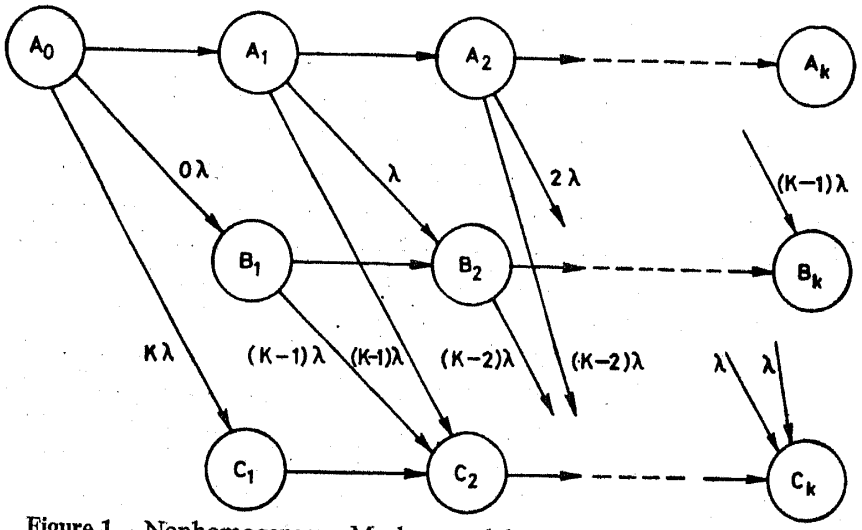

Figure 1. Nonhomogeneous Markov model

program redundancy, time redundancy, architecture reconfiguration and modular redundancy.

4.1 *Component redundancy*

This refers to component level redundancy (Moore & Shannon 1956) for both passive and active elements. Quadded redundancy is a popular scheme in this area (Teoste 1964; Kuehn 1969). Its main drawback for logic cricuits is that it reduces the noise margin.

4.2 *Program redundancy*

In FTC systems where the self test and reconfiguration is achieved through software, it is essential to keep certain programs stored in multiple copies (Bennetts 1978). This is equally true for critical application programs. This software redundancy works best against transients and is highly flexible. Additional storage, possibility of unrecoverable hardware failure in additional storage and lack of models to quantify software reliability are its disadvantages.

4.3 *Time redundancy*

In time redundancy, also known as roll-back and recovery, a program (at any level) is repeated to check the validity or otherwise of an error message (Bennets 1978). Check pointers are introduced in the program either at regular intervals or at selected optimal points. At each of these pointers the status data of the machine and other information necessary to restore the computer to a point where it was functioning perfectly, are stored in a standby memory. On detecting a fault the machine goes back to its previous known good state and starts its execution. This technique is ideally suited for transient faults. The disadvantage is that the check pointers are dependent on the application programs.

4.4 *Architectural reconfiguration*

When the system encounters a fault, it reconfigures into one of its acceptable degraded states of performance for accomplishing the critical functions. Thus this method makes a compromise between performance and reliability through architectural reconfiguration. The gracefully degradable computing system and the redundant system for safety fall into this category. Continuous system availability, system integrity and survivability are the major requirements of the present-day FTC systems for life critical control functions in aerospace applications (Hopkins 1977). Availability of a minimum level of computing power which accomplishes the user success criteria overshadows all other criteria. Modular multiprocessor system (MMPS) shown in figure 2 is one of the computing system architectures which can guarantee a minimum level of computing power, depending on the initial design, in spite of failures in different modules by pooling the resources and reconfiguring automatically to one of the worthwhile configuration with reduced computing capacity (Jacques Losq 1977). This type of system can effect a trade-off between redundancy and performance, on-line, depending on the phase of the mission i.e. high-reliability

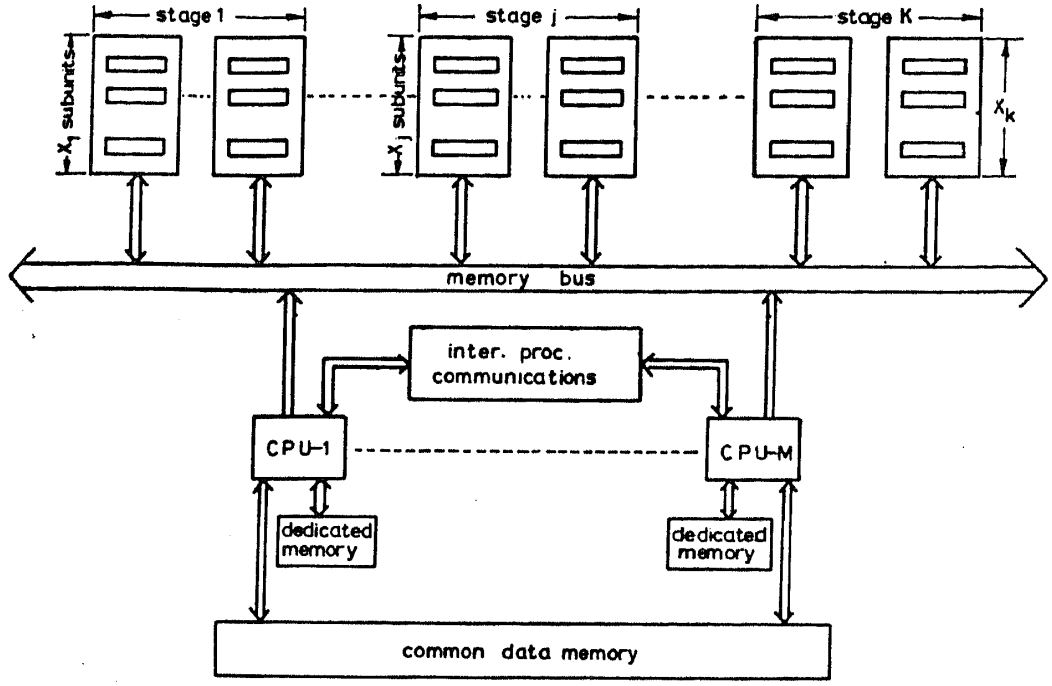

Figure 2. Schematic of a typical modular multiprocessor system

mode or high-computational mode or a dormant mode (Bricker 1973; Murray *et al* 1977). Some of the commercial gracefully degradable computing system examples are *prime, cmmp, cm*, cvmp, Pluribus* etc.; and those used in aerospace systems are STAR, COPRA, SERF, FTMP, SIFT, etc. There is another architecture known as bimodular redundancy (Courtois 1976) having safety as the main criterion where safety is defined as the probability of not sending nondetected incorrect data.

4.5 *Modular redundancy*

Modular redundancy uses more than one unit to continue correct operation in spite of faults. The basic modular redundancy techniques are static (fault-masking) dynamic and hybrid. Self-purging redundancy and reconfiguration schemes are other variations of the above. The survey papers (Short 1968; Su & Spillman 1977) give more details on the use of redundancy in digital systems. However, we restrict ourselves here to those redundant architectures that are commonly used in computing systems.

4.5a *Static redundancy* Any fault occurring in a static redundant scheme is instantaneously masked by taking the output of a majority voter till the time the required minimum redundancy level is not exhausted. However, the reliability of the voter introduces a new problem. The idea of a basic static redundancy scheme has been suggested by von Neumann (1956). The schematic diagram of a system with a basic static redundancy called triple modular redundancy (TMR) is given in figure 3. The output of each module passes through a majority voter whose output agrees with the majority of module outputs. N modular redundancy known as NMR is an extension of the basic TMR. Here there are N modules and it can tolerate $(N-1)/2$ module failures (figure 4).

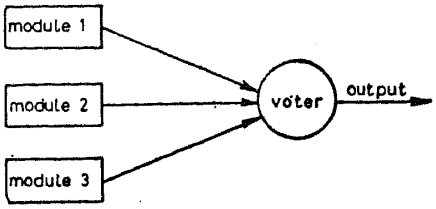

Figure 3. Triple modular redundancy

Static redundant schemes are effective against both transient and permanent faults and faults are masked instantaneously. Cumbersome fault detection and identification procedures are not necessary. The major drawback is that there is no warning when the system fails finally. Also the cost may be greater depending on the level of redundancy applied. The assumption of module independent failures is not valid at logic level within large scale integration (LSI) or medium scale integration (MSI). This scheme cannot mask multiple faults (Avizienis 1978).

4.5b *Dynamic redundancy* In dynamic-redundant systems, one unit operates and the others will be switched on upon the detection of the failure of the operating unit by the fault detection mechanism (figure 5). This scheme can tolerate failures upto the number of spare units used. The fault detection and recovery procedures used in such systems are more complicated and is discussed in detail in § 5. Module failure independence is valid when power switching is used.

4.5c *Hybrid redundancy* The advantages of both static and dynamic redundancy schemes are utilised in hybrid redundancy (Bricker 1973). This system consists of a static core with added spares. The fault masking feature of the static core is extended to include fault detection also and spares are added to replace the faulty module. The basic hybrid redundant system is the 'TMR+spares' system. As

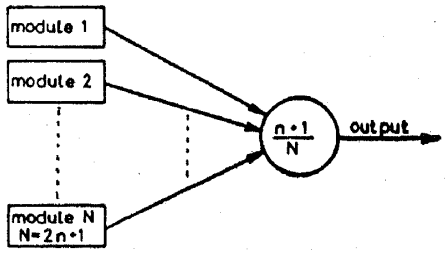

Figure 4. *N* modular redundancy

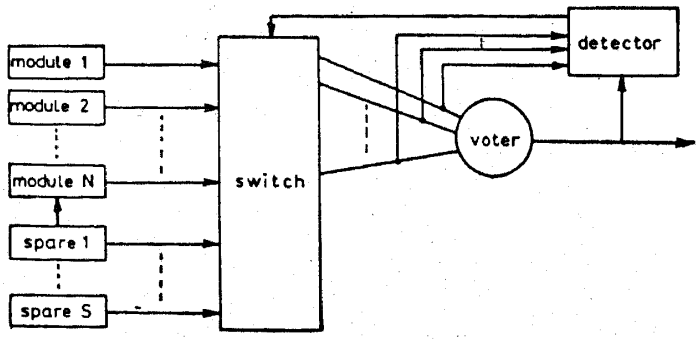

Figure 5. Hybrid (*N*, *S*) redundant system

and when the fault occurs, the spare will be switched into the system till it reaches the TMR core. Even though the complicated fault detection and identification mechanism of the dynamic system is avoided, the switching mechanism introduces complexity. The major disadvantage is that a multiple fault will lead to a system crash like that in the static scheme. System crash can be reduced by having an error detecting system added to each module (Ramamoorthy & Han 1975), which will increase the cost and complexity of the system. The other variation possible is to extend the static core and have 'hybrid NMR'. Mathur (1971a) has found that the gain in reliability will be greater if a new module is added to spare rather than to the basic core.

4.5d *Self-purging redundancy* Self-purging scheme (Jacques Losq 1976) overcomes the drawback of the complicated switch design. All the modules are connected to a threshold voter through a switch (figure 6). On detecting a failure in a module, that module's output is masked in such a way that it does not contribute to the vote taking in the threshold voter. This is easily achieved by making its output always to zero value. The self-purging scheme is again not protected against multiple faults. However, there are many other variations and one such design is suggested by Jacques Losq (1974).

4.5e *Reconfiguration scheme* Search for protection against multiple faults has led to another redundant scheme by Su & Ducasse (1975) known as the reconfiguration scheme. The basic system in this scheme starts with 5MR (NMR scheme with $N=5$). On a single failure, it reconfigures itself to a 'TMR+spare' i.e. hybrid scheme. On encountering a double fault it reconfigures itself to a simple TMR system. Also the switching scheme is very simple. It is possible to extend this scheme to a general NMR ($N>5$) reconfiguration scheme. But the switching mechanism will be complex.

4.6 *Self-checking*

Interplanetary space missions require an equally high computation capacity at the end of the mission as at the beginning. This demands that the coverage of any redundant scheme be nearly perfect. In order to achieve this nearly perfect coverage, Rennels (1978) suggests that each module in the redundant configuration should have a self-checking capability added to it.

4.7 *Coding techniques*

As is pointed out by Sousa (1976), combined techniques of information theory and fault-tolerance theory provide a means of achieving an FTC system, which uses

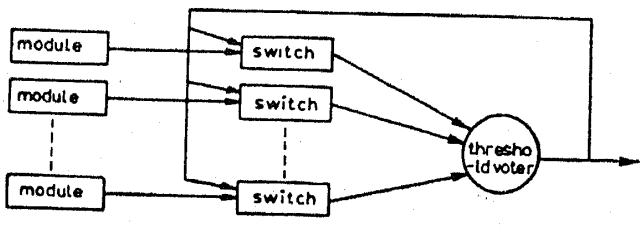

Figure 6. Self-purging redundant system

unreliable circuits to process the information which may or may not be unreliable due to some subsystem failure.

By coding the information, a mapping is introduced (Bennetts 1978) between 2^m input space to any one of the 2^n output space vectors. When there is no redundancy, any fault will produce an output which is valid but incorrect for that particular input combination. By expanding the output vector space from 2^n to 2^{n+s}, an internal fault will cause the output to lie outside the 2^n vector space but within 2^{n+s} space thereby indicating a fault (figure 7).

5. Coverage modelling

In the reliability modelling of a fault tolerant digital computing system, a factor coverage (c) (Bouricius *et al* 1969) is introduced to take care of the imperfectness in the detection, identification and recovery processes. The coverage c is defined as the conditional probability,

$$c = \Pr\{\text{system recovers} \mid \text{systems fails}\}.$$

Bouricius *et al* (1969) showed that even a value of $c=0{\cdot}99$ is nowhere near perfect coverage and stressed that the assumption of perfect coverage may lead to gross errors, which is obvious from the curves showing c vs I and c vs $1/I$. Also, increasing the level of redundancy has diminishing returns when the coverage is imperfect. Before going into the detailed mathematical analysis of coverage, it is better to understand the sequence of actions that normally take place in a fault-tolerant computer from the time of occurrence of a fault. To locate a fault there is a monitor. The monitor normally uses any one or a combination of the following methods such as hardware, software, repetition, coding, self-checking, reasonableness check and processing time check. The monitor can be classified according to the time of application as initial testing, concurrent testing (during the normal operation) and scheduled testing (off-line) (Avizienis 1978). Obviously, the fault detection is the most important factor in the coverage analysis as any further processing of a fault depends mostly on the effectiveness of fault detection. Before recovery action starts it is necessary to identify the faulty unit. Normally, the mechanism of identifying the faulty unit is combined with the detection mechanism. After the fault has

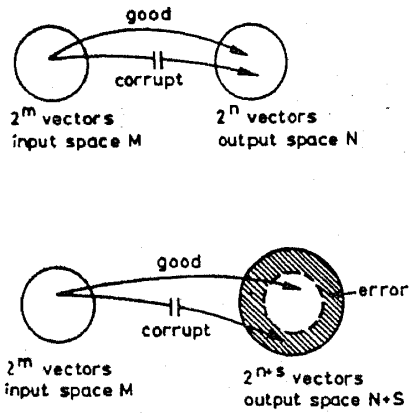

Figure 7. Redundancy using coding techniques for fault detection

been identified, the failed module has to be isolated from the system and a spare has to be replaced (in case of redundant spares). Then the system has to be brought to a previously known good state from where it can continue the operation in a normal manner. The design of fault recovery algorithms is basically dependent on the method of fault detection and identification. Two general categories of fault recovery mechanisms are manual and automatic. After recovery the system may return to an all good state (i.e.) full recovery or to a degraded state, depending on the architecture of the system used and available resources to be used in recovery.

5.1 *Modelling and analysis*

The fidelity with which any analytical model determines the reliability of an actual system is greatly dependent upon the accuracy with which the various coverage probabilities can be determined. Any analytic model for coverage therefore has to take into account factors such as type of failure experienced, its temporal characteristics (permanent or transient), the number of spares that must be tested before an operational one is found, time delays associated with various fault detection and recovery mechanisms, and the probability that a successful recovery can be achieved.

5.1a *Reliability modelling system (RMS) coverage model* When the concept of coverage was introduced, it was invariably assumed to be constant (Bouricius *et al* 1969; Bricker 1973). However, Rennels & Avizienis (1973) recognised that the coverage may vary because the fault detection may be state-dependent and the recovery procedure may be time-dependent based on the application requirements. Also the measurement of coverage is a tedious job. The coverage is broken down into its constituents. They are algorithmic coverage (A_c) which is defined as the conditional probability of detector and recovery algorithm, working correctly in effecting a recovery when there is a submodule failure and R_m which is the probability of recovery hardware not failing unrecoverably when there is a submodule failure. Algorithmic coverage A_c is further divided into two parts:

A_c^a —Pr {proper detection and recovery | fault in one of the active modules},

A_c^s —Pr {proper detection and recovery | fault in one of the spare modules}.

This coverage model is used in RMS (Rennels & Avizienis 1973) which is an iterative procedure for the reliability evaluation.

5.1b *Computer-aided reliability estimation-II (CARE II) coverage model* In CARE II (Stiffler 1975), the mutually exclusive fault classes of the computing system at different stages are identified. The effectiveness of different detectors to detect the faults in various classes of faults is computed. These two parameters are used to find the coverage coefficient of the stage

$$c_k = \sum_j d_j \sum_i c_{ijk},$$

where c_{ijk} = coverage associated with detector i for fault class j and d_j = fraction of total faults belonging to class j. The detailed computation of c_{ijk} can be found

$\bar{\lambda}_i$ = the mean failure rate of the total number M of permanent faults which cause an exit from i.

Then $r_{ij} = N/M.\ \bar{\lambda}_{ij} / \bar{\lambda}_i.$

Actually the ratio N/M is nothing but the fraction of covered failures and acts as a weighting factor. Therefore the coverage evaluation requires total possible permanent faults M, failure rate corresponding to each permanent fault and the effect of a fault. However, for digital computing systems, this becomes a formidable task, because of the lack of information about all failure rates, lack of knowledge about the fault sets M and N. Also the analysis of the effect of faults through failure mode and effect analysis (FMEA) grows enormously in size. However, General Electric Company has done this analysis through the statistical estimation approach under certain assumptions. The details can be found in Bjurman *et al* (1976).

5.1d *A specific case of mission phase related coverage study* Because a simple failure rate of a module and a constant coverage parameter may not be able to amply describe the different manifestations of monitor misbehaviour, O'Hern (1975) introduced a detailed study of monitor misbehaviour in redundant system monitor model (RSMM). The parameters that are used to describe a monitor model in addition to other failures are:

P_{oc} = probability of monitor failure to detect faults by comparison,

P_H = probability of monitor failure in a false alarm mode,

P_G = probability of monitor failure to identify the failed module by the single string test,

P_W = probability of a monitor failure to select the correct signal for transmission when the failure is undetected.

For each mission phase a transition matrix is formed with appropriate parameter values, which, when multiplied with the initial status of the redundant system states give the final state of the redundant system. Each matrix element of the above transition matrix is a function of the appropriate module failure and mission phase-dependent monitor failure probabilities

$$\begin{bmatrix}\text{final}\\ \text{redundant}\\ \text{system}\\ \text{states}\end{bmatrix} = \begin{bmatrix}\text{critical}\\ \text{phase}\end{bmatrix}\begin{bmatrix}\text{high}\\ \text{compu-}\\ \text{tation}\\ \text{phase}\end{bmatrix}\begin{bmatrix}\text{dormant}\\ \text{phase}\end{bmatrix}\begin{bmatrix}\text{high}\\ \text{relia-}\\ \text{bility}\\ \text{phase}\end{bmatrix}\begin{bmatrix}\text{initial}\\ \text{status of}\\ \text{redundant}\\ \text{system}\end{bmatrix}$$

6. Reliability modelling and analysis of fault-tolerant computing systems

It is mandatory for all manufacturers of complex electronic equipment manufacturers to furnish convincing reliability figures to the users. Reliability analysis of

$\bar{\lambda}_i$ = the mean failure rate of the total number M of permanent faults which cause an exit from i.

Then $r_{ij} = N/M.\ \bar{\lambda}_{ij} / \bar{\lambda}_i.$

Actually the ratio N/M is nothing but the fraction of covered failures and acts as a weighting factor. Therefore the coverage evaluation requires total possible permanent faults M, failure rate corresponding to each permanent fault and the effect of a fault. However, for digital computing systems, this becomes a formidable task, because of the lack of information about all failure rates, lack of knowledge about the fault sets M and N. Also the analysis of the effect of faults through failure mode and effect analysis (FMEA) grows enormously in size. However, General Electric Company has done this analysis through the statistical estimation approach under certain assumptions. The details can be found in Bjurman *et al* (1976).

5.1d *A specific case of mission phase related coverage study* Because a simple failure rate of a module and a constant coverage parameter may not be able to amply describe the different manifestations of monitor misbehaviour, O'Hern (1975) introduced a detailed study of monitor misbehaviour in redundant system monitor model (RSMM). The parameters that are used to describe a monitor model in addition to other failures are:

P_{oc} = probability of monitor failure to detect faults by comparison,

P_H = probability of monitor failure in a false alarm mode,

P_G = probability of monitor failure to identify the failed module by the single string test,

P_W = probability of a monitor failure to select the correct signal for transmission when the failure is undetected.

For each mission phase a transition matrix is formed with appropriate parameter values, which, when multiplied with the initial status of the redundant system states give the final state of the redundant system. Each matrix element of the above transition matrix is a function of the appropriate module failure and mission phase-dependent monitor failure probabilities

$$\begin{bmatrix}\text{final}\\ \text{redundant}\\ \text{system}\\ \text{states}\end{bmatrix} = \begin{bmatrix}\text{critical}\\ \text{phase}\end{bmatrix}\begin{bmatrix}\text{high}\\ \text{compu-}\\ \text{tation}\\ \text{phase}\end{bmatrix}\begin{bmatrix}\text{dormant}\\ \text{phase}\end{bmatrix}\begin{bmatrix}\text{high}\\ \text{relia-}\\ \text{bility}\\ \text{phase}\end{bmatrix}\begin{bmatrix}\text{initial}\\ \text{status of}\\ \text{redundant}\\ \text{system}\end{bmatrix}$$

6. Reliability modelling and analysis of fault-tolerant computing systems

It is mandatory for all manufacturers of complex electronic equipment manufacturers to furnish convincing reliability figures to the users. Reliability analysis of

a conventional digital computing system is not difficult because the computer is considered failed as soon as a component fails. However, high reliability and fault tolerance requirements make the reliability estimation of the fault-tolerant computing system difficult. In general, the reliability of any fault-tolerant computing system can be estimated through hardware monitors, simulation and mathematical modelling or any combination of the above (Ng 1976).

The first approach is costly and requires a long time to collect sufficient statistical data. It becomes all the more difficult as the complexity of the system increases. Practical limitation of the speed and size of even the largest computer available today presents severe restrictions on the extent of simulation accuracy of complex FTC systems, in addition to the large computer time and money required. The third approach, the mathematical modelling, is a conceptual abstraction of the real system, and is easily amenable to analysis by established mathematical tools. However, the accuracy of the analysis of the complex system is brought down by the assumptions that have to be made to bring down the model to a mathematically tractable one. But the cost of the analysis is very low and it is highly flexible.

6.1 *Combined hardware-software failure analysis*

Combined analysis of both hardware and software originated failures may be necessary as their effects are often indistinguishable. Landrault & Laprie (1977) present such an analysis combining the software failure model proposed by Trivedi & Shooman (1974) with a hardware failure model. The system here is subjected to a hardware failure process with a constant failure rate and a software failure process with constant failure rate for a given number of errors. The set of states that the system can occupy is given in figure 9 and they are

a, state i^1 $\{i \in (1, 2, \ldots, n+1)\}$
—operating states after $i-1$ repair actions following a software failure,

b, state i^2 $\{i \in (1, 2, \ldots, n+1)\}$
—repair state after a hardware failure from i^1, and

c, state i^3 $\{i \in (1, 2, \ldots, n)\}$
—repair state after a software failure.

The above model is used to find the availability of the computer system (both redundant and nonredundant) using the Laplace-Stieltjes transform technique.

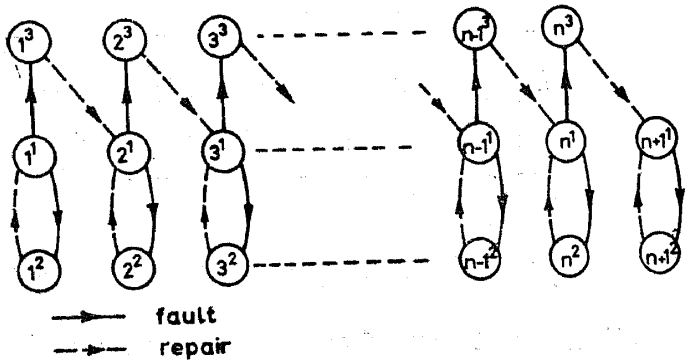

Figure 9. Combined hardware-software failure model

6.2 *Need for automated reliability modelling*

To have a first hand idea about the scheme to be selected among the innumerably available ones and for further comparison, the designer needs an automated procedure because the number of parameters describing the redundant scheme and their variations are on the increase.

6.2a *CARE* (Mathur 1971b) is one such automated interactive computer program which can 'model self repair and fault-tolerant computer organisations, compute reliability theoretic functions, perform sensitivity analysis, compare competitive systems with respect to various measures and facilitate report preparation by generating tables and graphs'. CARE has a repository of equations describing various basic redundancy schemes. These basic equations can be combined in any proper manner to model a complicated system. Some of the important measures available as the output of the CARE are system mean life, reliability at mean life, gain in reliability compared to some other configuration, reliability improvement factor and the mission time availability for some specified reliability.

6.2b *Reliability modelling systems* Reliability modelling system (RMS) (Rennels & Avizienis 1973) is an extension of CARE in the sense that the state-dependent and time-varying nature of coverage is taken into account and a recursive solution is formed for systems with a limited number of spares.

$$R^*(N, S, A_c)[t] = \exp(-N\lambda t)\exp(-S\mu t) + (A_c^a\lambda N + A_c^s S\mu)$$
$$\int_o^t \exp[-(N\lambda + S\mu)X]\, R^*(N, S-1, A_c)[t-X]\, dX$$
$$+ \sum_{i=1}^{S}(1-A_c^s)\int_o^t \exp[-(N\lambda+S\mu)X]R^*(N, i-1, A_c)[t-X]dX.$$

RMS is implemented in APL as an interactive system to predict the reliability of dynamic redundant computer systems.

6.2c *CARE II* implemented by Raytheon Company (Stiffler 1975) can be easily explained with the diagram in figure 10. The fault-tolerant computing system is divided into $n(1 \leqslant n \leqslant 8)$ stages, with switchable spares for each stage. The two modes of possible operation are specified by mode 1 and mode 2. Mode 1 and mode 2 needs Q_{1i} and Q_{2i} identical functioning units respectively, at any state i for the system to be operational. A failure in mode 1 forces the system to mode 2 by exhaustion of spares or to the failed state when there is an uncovered or a single point failure. The system continues its operation in mode 2 until a coverage failure, single point failure or exhaustion of spares causes the whole system to fail. CARE II by its nature covers both permanent and transient faults. Also it includes a detailed coverage model which can evaluate the coverage coefficients for each stage of the system. This has already been described in § 5.

6.3 *Unified approach for reliability modelling and analysis*

The reliability modelling and analysis of new systems as well as of small variations of existing systems were too difficult because of the fact that the earlier analyses were aimed only at a particular system configuration and no attention was given to interesting variations of the same (Ng 1976). This necessitated a unified approach,

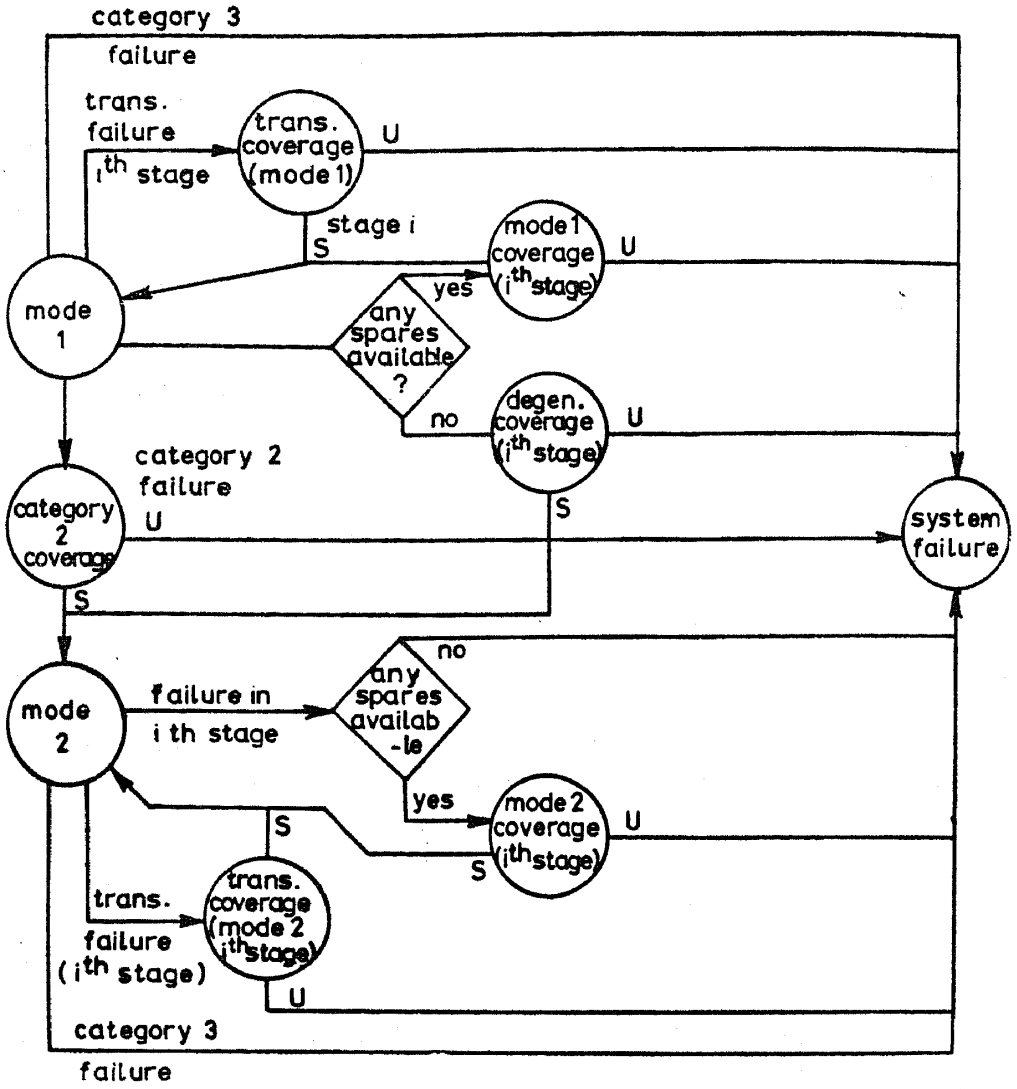

Figure 10. CARE II reliability model

6.3a *Five-parameter hybrid redundancy* The first approach in this direction is that of Bricker (1973) who shows that the three-parameter hybrid degraded redundancy denoted by $H\{N, S, D\}$ universally describes both standby and the classic NMR structures. Here N need not be an odd integer. D describes the minimum number of units that must be operative for the system to be in degraded mode. Bricker also introduces a five-parameter model $H\{N, S, D_1, D_2, D\}$. The system works as $H\{N, S\}$ until D_1 active units are left, at which point D_1-D_2 of these are discarded; the system now containing D_2 active units works until the system has only $D-1$ active units left. Even though this model tried to unify different schemes by the above five-parameter hybrid degraded model, the assumption of perfect detection, identification and recovery of fault ($c=1$) made this model unpopular.

6.3b *General modular redundancy (GMR)* GMR is an attempt at universal analytical treatment for modular redundant architectures (Sousa 1976). It provides a unified reliability modelling and an enlarged perspective of redundancy theory. The reliability analysis of GMR structure is so general that it can be a function of many parameters other than N, M and S (i.e.) $G\{N, M, S, P_0, P_1, P_x, c, R_r, \rho\}$. When

the reliability of a particular GMR case is not a function of a given keyword parameter, this parameter assumes a default value. Figure 11 illustrates how this unifying notation describes the various system configurations covered by GMR. A recursive technique is used to evaluate the reliability for $G\{N, M, S\}$. This is again determined from the summation of probabilities of the events that lead to system success:

(1) all channels survive mission time $T \rightarrow R_0^N R_S^S$,

(2) spare is the first to fail (i.e.)

$$G\{N, M, S\} \rightarrow G\{N, M, S-1\} \rightarrow Sc_d \int_0^T \exp(-N\lambda\tau)\,\mu \exp(-\mu\tau) \exp[-(S-1)\mu\tau]\, R(N, M, S-1)\,[T-\tau]\, d\tau.$$

where $R(N, M, S-1)$ is the reliability of $G\{N, M, S-1\}$ structure and

(3) active channel fails first

$$\rightarrow Nc_a \int_0^T \lambda \exp(-\lambda\tau) \exp[-(N-1)\lambda\tau] \exp(-S\mu\tau)\, R(N, M, S-1)\,[T-\tau]\, d\tau.$$

Therefore $R(N, M, S)[T] = R_0^N R_S^S + (Nc_a\lambda + Sc_d\mu) R_0^N R_S^S \int_0^T \exp(-N\lambda t) \exp(-S\mu t)\, R(N, M, S-1)[t]\, dt.$

This recursive equation has a closed form solution

$$R(N, M, S)[T] = \sum_{j=0}^{N} B_j^S R_0^j + \sum_{i=1}^{S} B_{N+i}^S R_0^N R_S^i,$$

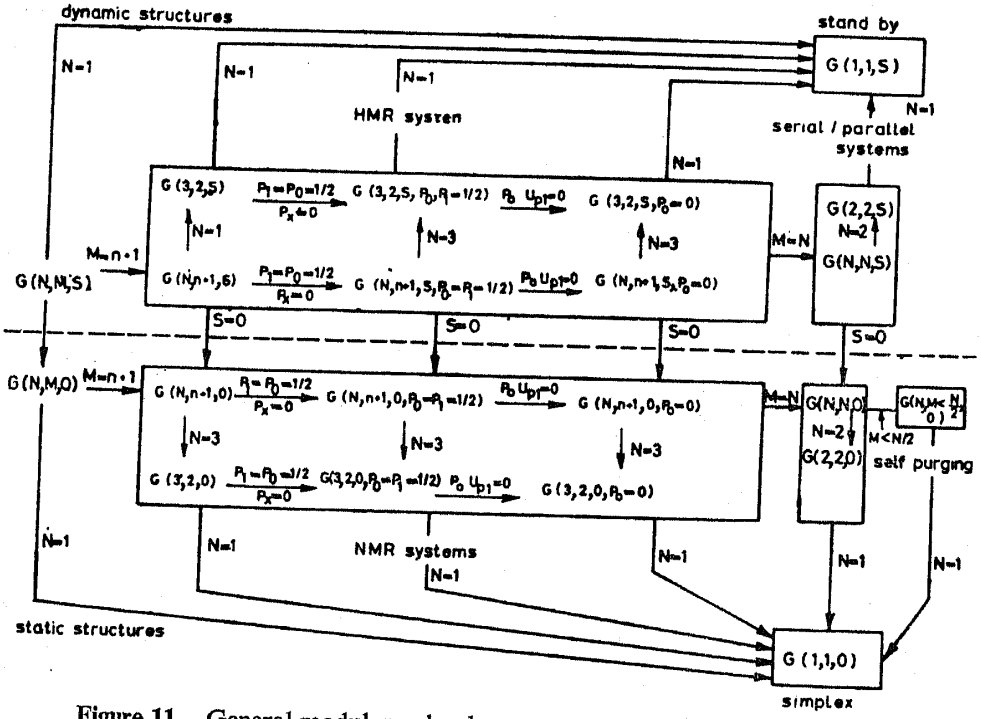

Figure 11. General modular redundancy

where
$$B_j^S = \frac{\binom{LAS}{S}}{\binom{(N-i)/(\delta+S)}{S}} (c_d)^S B_j^0;\ j=0, 1, .., N,$$

$$B_{N+i}^S = (c_d)^{S-i} \binom{LAS}{S-i} \left\{ \sum_{l=0}^{i-1} \binom{LA+i}{l} (-C_d)^l - c_d^i \sum_{j=0}^{N} \left[\sum_{l=1}^{i} (-1)^{i-1} \frac{\binom{LA+i}{i-l}\binom{LA+l}{l}}{\binom{(N-j)/(\delta+l)}{l}} \right] B_i^0 \right\}$$

$$i = 1, 2, .., S;\ S \geqslant 1.$$

6.3c *Markov reliability models for closed FTC system* The reliability modelling of FTC systems assumes that the first failure in a module results in the failure of that module. In general all failed states are aggregated into a single absorbing state. However, the operational status is usually characterised by a set of states finite in number each of which represents a useful working state of the system. The occurrence of a failure causes a transition from one state to another in the model resulting in a reconfiguration of the system. The transition between states is governed by two processes; a failure process followed by a recovery process. The combined process of failure and recovery is memoryless. The reliability model for a closed FTC system can be represented by a finite state continuous time homogeneous Markov model (Ng 1976).

As there is no repair in closed FTC systems, the Markov reliability model represents a ' death ' process, i.e., the system moves into successively smaller configurations until an absorbing state is reached. In this modelling the state number represents the number of nonfailed modules and hence transition from j to i is impossible if $i > j$. There are one or more possible transitions out of that state to lower-indexed states. Let the given state be i and let σ_{ij} be the transition probability from state i to j

$$\sigma_i = \sum_{\substack{all\ i \\ j<i}} \sigma_{ij},$$

is the unconditional probability of transition out of state i. Clearly σ_i is the failure rate of the subsystem at state i. Let $P_i(t) = P_r\{$system is in state i at time $t\}$. Then the system can be described by a system of differential equations

$$\dot{P}_i(t) = (p_{ii} - 1)P_i(t) + \sum_{j \neq i} p_{ji} P_j(t).$$

Let
$$\bar{Q}(t) = [P_1(t), P_2(t), \ldots, P_n(t)],$$

and
$$A = \begin{bmatrix} a_{11} \ldots a_{1n} \\ \vdots \\ a_{n1} \ldots a_{nn} \end{bmatrix},$$

where $a_{ij} = p_{ji}$ and $a_{ii} = p_{ii} - 1$. As we have assumed that there is only one absorbing state, A can be represented as

$$\left[\begin{array}{c|c} B & \begin{matrix}0\\0\\\vdots\end{matrix} \\ \hline a_{n1} \cdots a_{n,\,n-1} & 0 \end{array}\right].$$

If $\bar{P}(t) = [P_1(t), \ldots P_{n-1}(t)]$, the solution is $\bar{P}^T(t) = \exp(Bt)\bar{P}^T(0)$.

Since the eigenvalues of the closed FTC systems are given by $-\sigma_i$, $i=1,\ldots m$, are all distinct, we have a more explicit solution for $\bar{P}(t)$. Ng has shown that

$$R(t) = \sum_i A_i \exp(-\sigma_i t) \text{ over all good states.}$$

Also $$\text{MTTF} = \sum_i \frac{A_i}{\sigma_i}.$$

Ng (1976) has given a reliability model for a general closed FTC system that is not constrained in its architecture. The state diagram of an FTC which allows degradation of active configuration at computer level is given in figure 12. Its model is represented by a set of parameters $(N, D, S, c_a, c_d, \lambda, \mu, \mathbf{Y}, \mathbf{CY}$ without any constraints. The selection of spares occurs in a linear order. The effect of unrecoverable failure is incorporated in the model by transitions to the states with an overbar $\overline{(N, S-1)}$ $\overline{(N, 0)}$ etc which cannot degrade the active configuration, i.e., states (N, i) and $\overline{(N, i)}$ have the same effective configuration, but the system at state $\overline{(N, i)}$ has lost

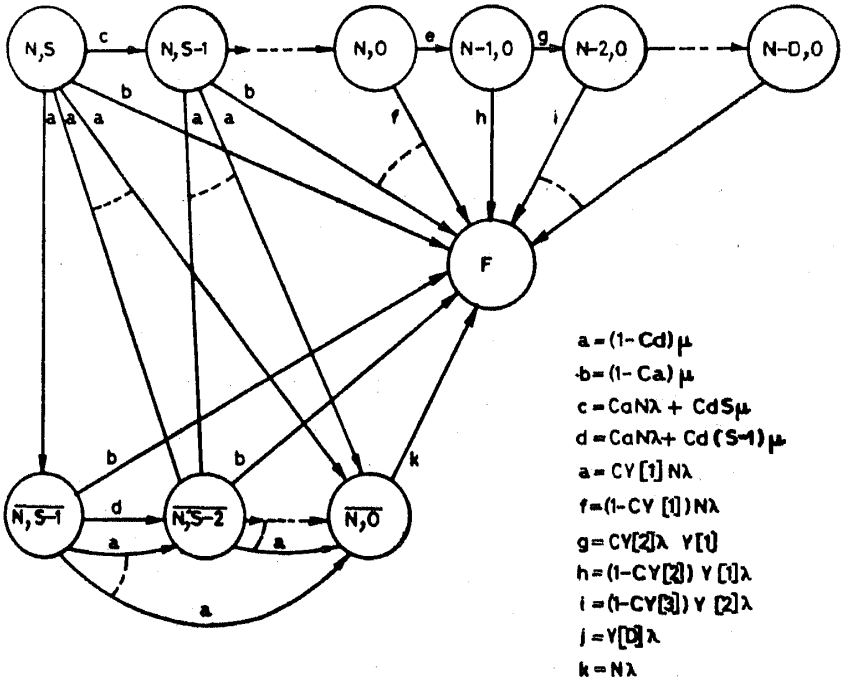

Figure 12. ARIES—General Markov reliability model

its ability to 'gracefully degrade' because of an unrecoverable failure in one of its spare modules. The reliability equation of this general model is of the following form

$$R(t) = X(t)\, A \cdot \mathrm{W}(t),$$

where $$X(t) = [\exp(-Y[0]\lambda t), \exp(-Y[1]\lambda t), \ldots, \exp(-Y[D]\lambda t],$$

$$\mathrm{W}(t) = [1, \exp(-\mu t), \ldots, \exp(-S\mu t)],$$

and $$A = \begin{bmatrix} A^0_{S,0} \ldots . A^0_{S,S} \\ \cdot \quad\quad \cdot \\ \cdot \quad\quad \cdot \\ A^D_{S,0} \ldots . A^D_{S,S} \end{bmatrix}.$$

Matrix A is a sparse matrix. All its entries outside the first row and first column are zero. The coefficients $A^k_{s,j}$ in the matrix A are functions of the parameters and is computed by an iterative procedure. This equation represents a general and computationally efficient solution to the reliability analysis problem for FTC subsystem. This model is further extended to include repair and transient failures.

6.4 *Other specific methods of reliability analysis*

6.4a *Complementary analytic-simulative technique (CAST)* The complementary analytic-simulative technique (CAST) (Conn *et al* 1977) came out as a result of the fact that neither analysis nor simulation alone could satisfy the evaluation requirements of complex computer systems. Analytic modelling provides flexibility and rapid, economical data generation. Simulation permits computer system details to be included easily, but extensive data generation is slow and expensive. CAST permits the user to obtain the best features of both analytic modelling and simulation. CAST is very useful where there is a need for accurately modelling complex computer systems. This is achieved through the use of analytic models which utilise parameters determined both by simulation and engineering characterisation of the computer system (figure 13).

Analytic modelling has resulted in the successful formulation of survivability expressions for a reconfigurable computing system (RCS) composed of N computers, where N is any finite, positive integer. This model includes explicit

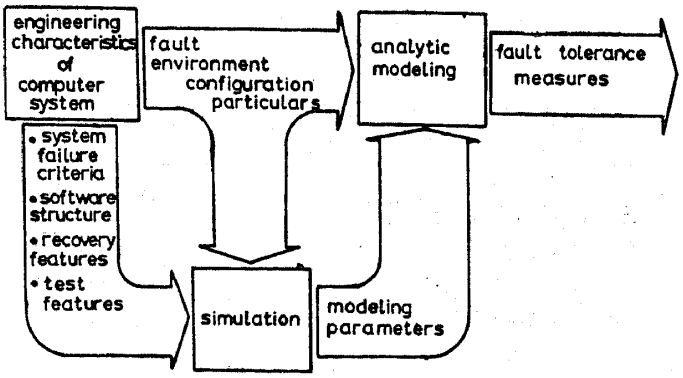

Figure 13. Complementary analytic-simulative technique (CAST)

consideration of components of coverage and the effects of transient faults. Hardware/software interactions are also implicitly included. The simulator portion of CAST is based on descriptions of the states the RCS could assume and the transitions between these states. Use of this approach results in a fault-driven simulation which minimises the simulation cost.

However CAST is applicable to multicomputer systems at computer levels only and does not go down to a subsystem level of analysis. Also when the number of states becomes enormous this approach becomes time-consuming.

6.4b *CARSRA* Computer-aided redundant system reliability analysis (CARSRA) (Bjurman *et al* 1976) is a general purpose reliability analysis program that handles modular redundant reconfigurable systems taking into account such factors as coverage and transient faults. It evaluates functional readiness and system failure probabilities for two different failure modes, using a unique approach that combines Markov modelling with dependency tabulation. As a result, the dimensional problem of the Markov model is overcome by partitioning the system into stages, or sets of redundant identical modules, so that each stage may be modelled by a dedicated Markov model of low order. For reliability analysis, the system is partitioned into stages and modules, where a module is defined as a set of elements performing a specified function. Each stage is modelled by a Markov model describing the different redundancy states of the stage. The last two states of figure 14 are failed states. The maximum number of states in the Markov model for each stage is limited to 9. The system dependency structure is described by a dependency tree diagram (figure 15) and a dependency table is formed by which the system dependency structure is specified for CARSRA.

In CARSRA the system failure probability $FP(t, \gamma)$ at a certain time γ from the beginning of the critical mission phase, is computed as:

$$FP(t, \gamma) = \frac{\sum_{i=1}^{n} FR_i(t) \cdot FP_i(\gamma)}{FR(t)}$$

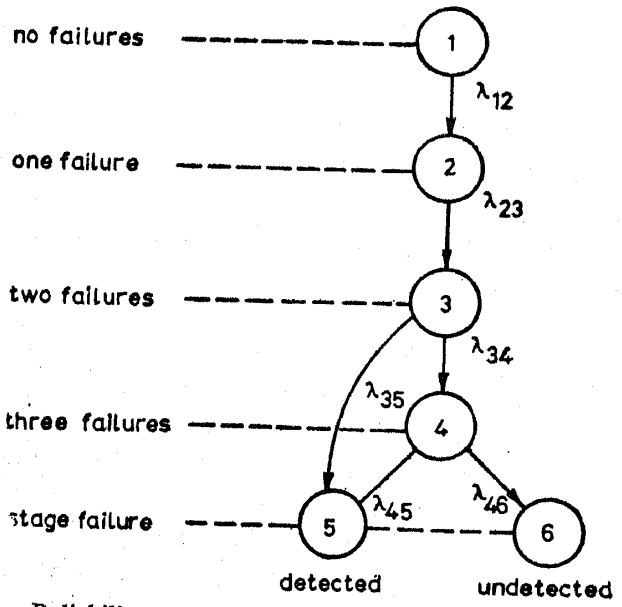

Figure 14. Reliability Markov model used in CARSRA

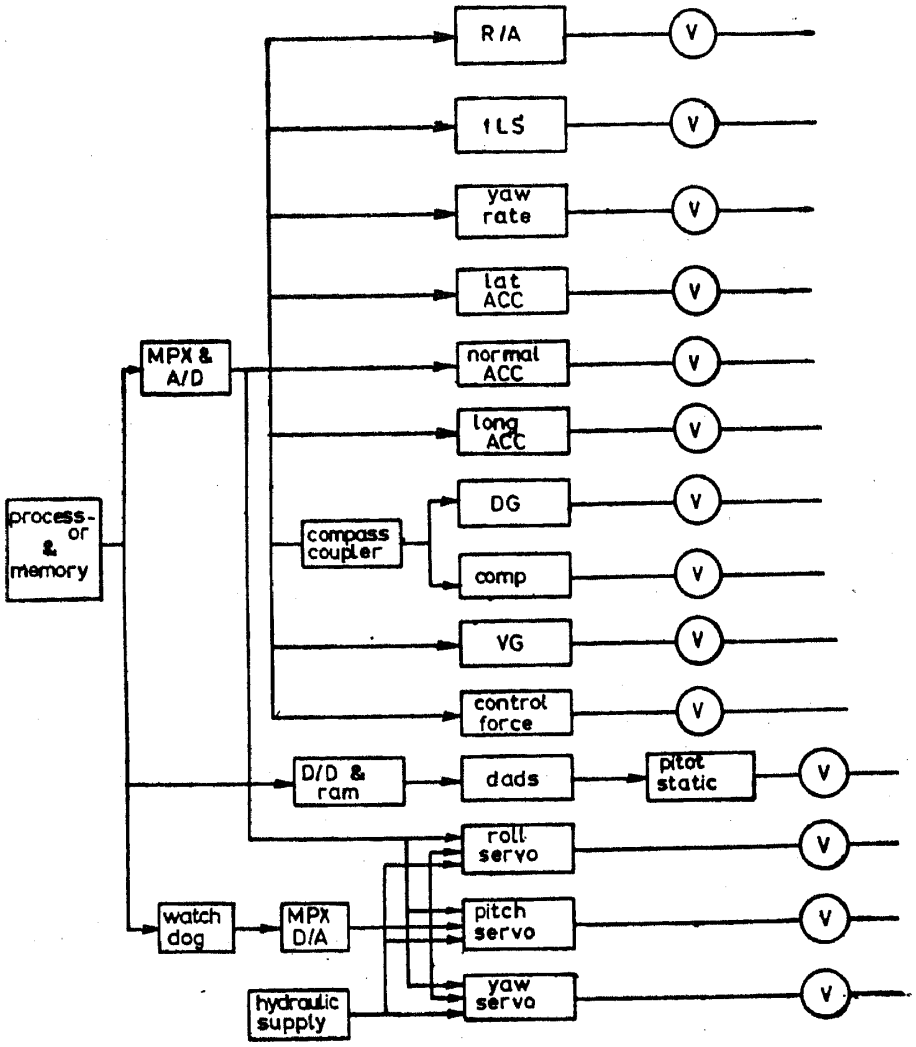

Figure 15. Dependency tree of a typical flight control system

where $FR_i(t)$ is the probability of having different module failure patterns and $FR(t) = \sum_{i=1}^{n} FR_i(t)$.

The structure of reliability estimation in CARSRA is as follows. The system is partitioned into stages and a dependency table is formed from a dependency tree and the Markov model is described for each stage. Now the failure rate and coverage analysis is carried out. Using all these data, the functional readiness and failure probability is computed.

6.4c *FTMP reliability analysis* The fault-tolerant multiprocessor (FTMP) system has different failure modes. Hopkins *et al* (1978) have shown that these failure modes are better modelled each with a different mathematical tool. Particularly, the probability of failure due to exhaustion of spares can be adequately modelled by combinatorial methods, whereas Markov processes are better suited for coverage-related problems. This type of decomposition of reliability modelling is feasible because each of these failure modes predominates in a different time segment. The final output of this analysis is the availability and survival probability.

(i) *Lack of coverage* Coverage is most conveniently modelled using the Markov process. The number of states in the Markov model is reduced to a minimum by assuming (a) the system is as good as new after recovery, (b) constant failure rate and (c) the system starts at all good state. The Markov model is given in figure 16 along with a list of all the catastrophic double failure combinations. The failure probability due to lack of coverage is seen to be a linear function of time. Since the total leakage rate is only about 10^{-9} per hour, the state probabilities diminish extremely slowly and a state of equilibrium would hold for hundreds of hours.
(ii) *Exhaustion of spares* To compute the probability of not having sufficient spares, it is necessary to define the minimum spares needed to operate successfully which is denoted as the critical minimum complement (CMC). This is mission-dependent as well as architecture-dependent.
(iii) *Bus guardian units (BGU) failures* These failures are again modelled by combinatorial analysis. Also this does not contribute significantly to the failure modes.

From figure 17 it is seen that the lack of a coverage model is best suited in the short run (0 to 50 hr) and lack of sufficient spares poses a problem only when the mission time is greater than 100 hr.

6.5 *FTA applications to computing system reliability modelling*

Using fault-tree analysis (FTA) we find out how a system fails. From this analysis, failure combinations which otherwise might not have been recognised can be unearthed.

Fault-tree representation can be considered as a systematic management tool to express in detail the system hierarchy. A basic event can be extended as another fault tree at a lower level. As an approach to safety and reliability, FTA is extremely

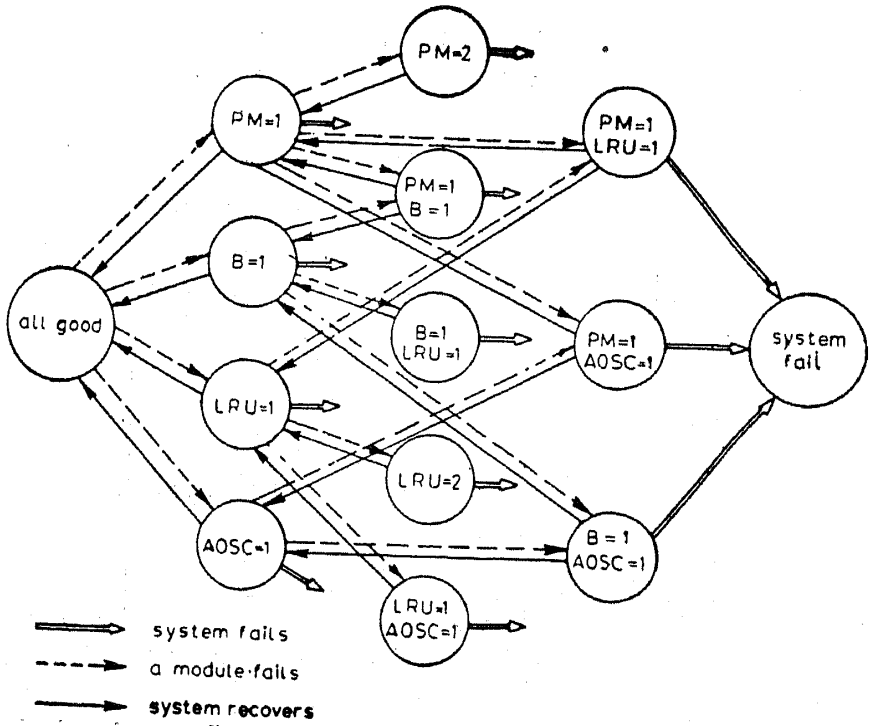

Figure 16. Lack of coverage Markov model used in FTMP reliability analysis

powerful. The analysis of computer system reliability shares common problems with the analysis of the other systems' reliability to which FTA has been applied successfully. Goldberg (1974) has studied the possible application of FTA to hardware faults in FTC systems.

Ramamoorthy *et al* (1977) applied the FTA approach to analyse the combined hardware and software failure effects in computing systems. Also they suggested the use of FTA for the performance analysis of computing systems. Goldberg also suggested the possible use of FTA for the coverage modelling of computing systems. This approach has been used in the reliability analysis of the COPRA (Brouaeys & Muron 1977) computer system. Here the COPRA system is partitioned into cells (failure of any component in a given cell will have the same effect on the computer). The event trees are used to find the probability of each cell going to any one of the defined states due to different failures (figure 18). These failure rates, obtained as above, are used in the Markov model of the COPRA computer systems. A similar approach is used for the coverage model of ARCS (Bjurman *et al* 1976).

7. Performability

Several multiprocessor system configurations have been proposed to meet the stringent reliability and performance requirements of aerospace computer systems (Hopkins *et al* 1978; Brouaeys & Muron 1977; Koczela & Burnet 1968). These systems have the ability to gracefully degrade. In such cases, the performance and reliability issues must be dealt with simultaneously in the process of evaluating system effectiveness (Holick 1963; Dorrough 1971; Meyer 1978). Conventional indicators of system reliability are reliability, MTTF, availability and MTTR. Ingle & Siewiorek (1977) consider the evaluation of these reliability measures for two multiprocessor system configurations (*Cmmp*, C_m*). On the other hand, Strecker (1970) and Bhandarkar (1975) consider a purely performance related issue—the memory interference problem in multiprocessor systems. System quality assessment should somehow combine both these aspects and Dorrough (1971) proposes a performance

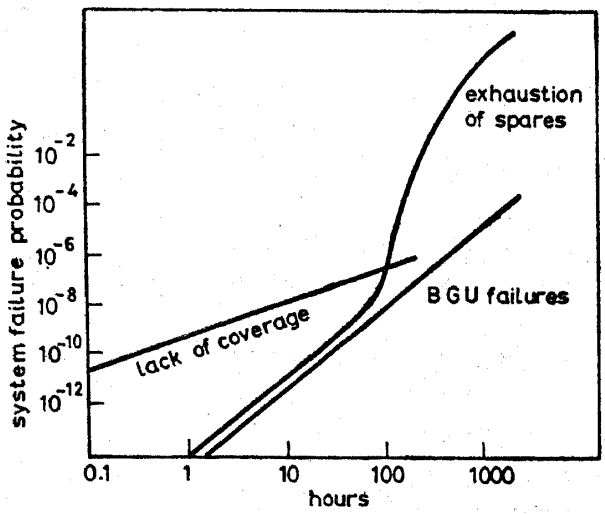

Figure 17. System failure probability for different failure modes of FTMP

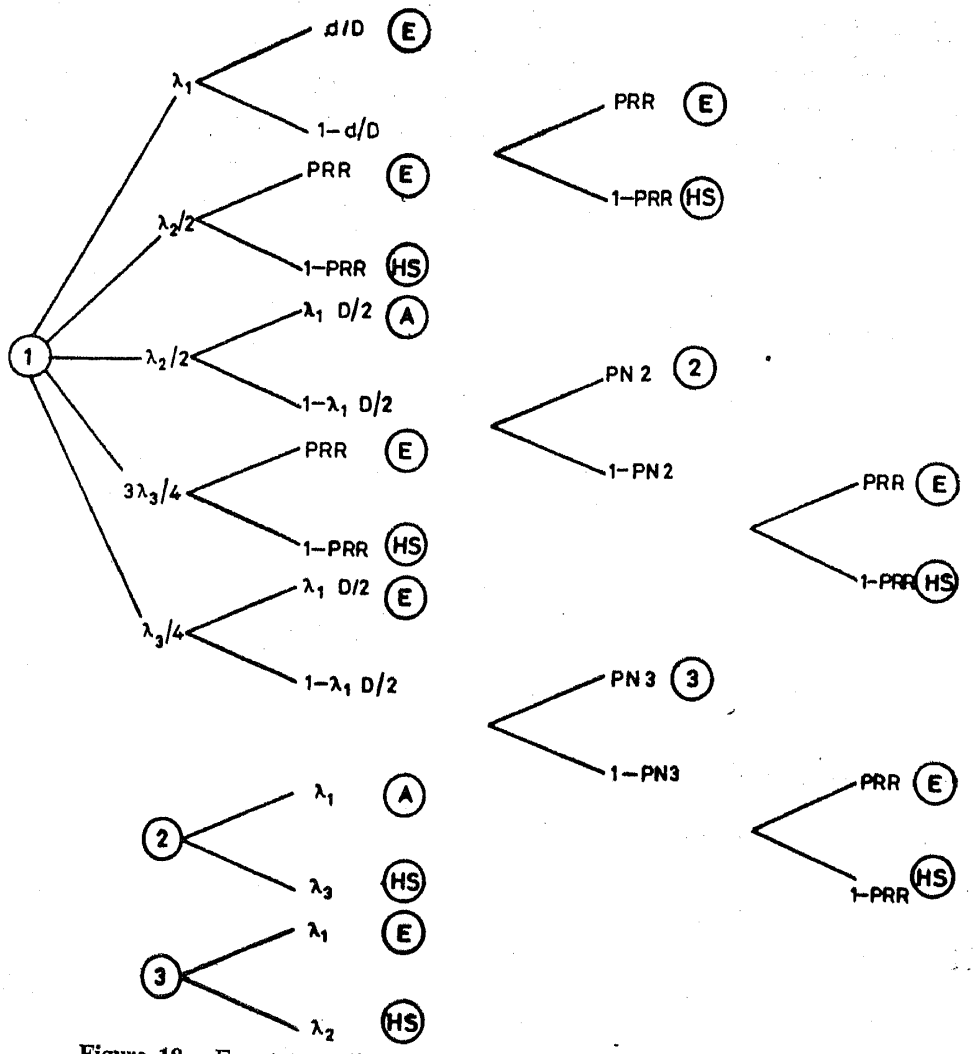

Figure 18. Event tree diagram for processors of COPRA computer

capability measure (PCM) as a real measure of system quality. Many such PCM analyses have been done in recent years with respect to gracefully degradable computing systems. All these studies aim at combining reliability and performance. Beaudry (1977) gives a number of computation-related measures, such as computation reliability, the mean computation before failure, and the computer availability and evaluates them for a multicomputer system using a transformed Markov model. Bellon & Saucier (1976) introduce normalised measures such as '*n* reliability' and '*n* availability' for the system effectiveness of a multiprocessor system. Troy (1977) computes the expected work power using graph techniques for a general computing system having a redundant structure for critical functions and the rest forming a high computational structure for the noncritical high computational requirements. Jacques Losq (1977) finds the availability of a gracefully degradable multiprocessor system by modelling the individual resources by a Markov model. He partitions the gracefully degradable computer system into its low level resources (processors, memory modules, interconnection links and I/O modules), and assumes that the amount of service that can be obtained from such a resource is proportional to the number of fault-free elements in that class of resources, which is almost similar to the '*n* availability' concept of Bellon applied to low level resources.

From Hoogendoorn (1977), the instruction execution rate can be written as IER$=Z/t_C$ where Z is the expected number of busy memories.

Therefore $P_C(i,j) = [Z(i,j) / t_C] / [Z(x,y) / t_C]$,

$$= Z(i,j) / Z(x,y),$$

$$= \frac{\text{expected number of busy memories at node } (i,j)}{\text{expected number of busy memories at node } (x,y)}.$$

Expected number of busy memories at any node (i,j) can be easily computed using Strecker's approximate closed form solution (Strecker 1970), and is given by

$$\sum_{k=1}^{j} P_r\{M_p(k) \text{ is busy}\},$$

$$= [j(1-(1-1/j)^t].$$

Therefore the normalised computing power at node (i,j) is given by:

$$P_C(i,j) = \frac{j[1-(1-1/j)^t]}{y[1-(1-1/y)^x]}$$

7.2 *System quality measure*

In a gracefully degradable computing system the main parameter of interest is the one which describes its ability to provide continuous service irrespective of the occurrence of failures. Computation availability is the apt parameter for quantifying the performance measure of the gracefully degradable computing system. The computation availability measure can be defined as the expected value of the computation capacity of the system at time t and it is denoted by

$$A_C(t) = \sum_{i,j} P_C(i,j) P_r\{i,j;t\}.$$

The values of $A_C(t)$ for a typical multiprocessor configuration with respect to the normalised parameter λT is plotted in figure 19.

The next parameter which describes such a gracefully degradable computing system is performability which quantifies simultaneously all the three parameters of interest (i.e.) performance, reliability and user-success criteria. In real-time applications like aerospace computing system, it is essential to have a certain length of computation at a prescribed throughput rate. When the throughput rate goes below a certain level, the computation takes a longer time and sometimes it may be disastrous in aerospace applications. Performability is given by:

$$R(t,T \mid a \geqslant a_{\min}) = \sum_{i,j} C(i,j;T) P_r\{i,j;t\},$$

where $C(i,j;T)$ is the capacity distribution,

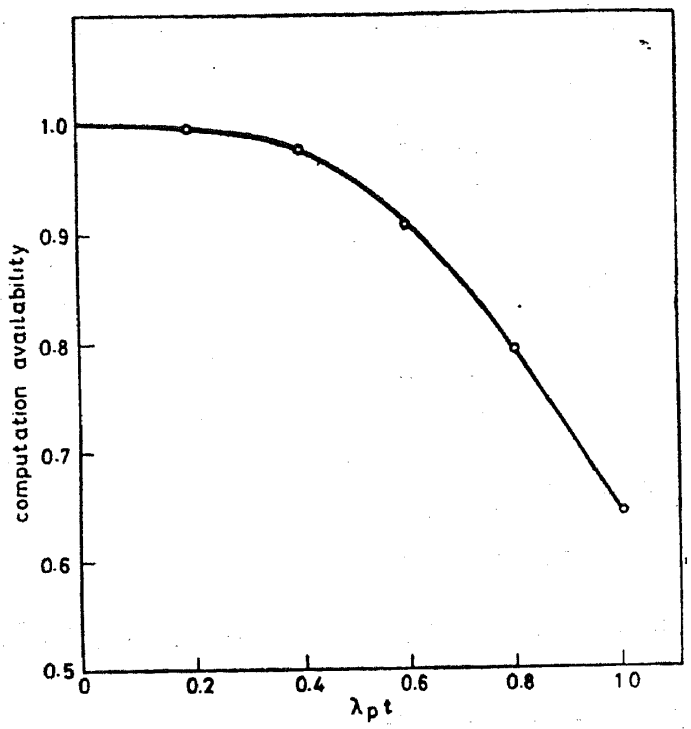

Figure 19. Computation availability

$$C(i, j; T) = P_r\{\text{system executes a task of length } T \text{ at a computing power} \geqslant a_{\min} \mid \text{state is } (i, j) \text{ at the beginning of computation}\}$$

$$= \sum_{\substack{g, h \\ a \geqslant a_{\min}}} P_r\{g, h; T\}$$

where $P_r\{g, h; T\} = P_r\{\text{system is in state } (g, h) \text{ after a computation of length } T \text{ at a computation power level } a \geqslant a_{\min}\}$.

Another parameter which describes the gracefully degradable computing system is system effectiveness, which is given by

$$E_{ff}(s) = \sum_{\substack{i, j \\ a \geqslant a_{\min}}} W(a)\, R(t, T \mid a \geqslant a_{\min}),$$

where $W(a)$ is the worth function. Here $W(a)$ takes the values of

$$W(a) = \begin{cases} 1 \text{ for } a \geqslant a_{\min}, \\ 0 \text{ for } a < a_{\min}. \end{cases}$$

For evaluating the performability $R(t, T \mid a \geqslant a_{\min})$, we have considered one phase of a mission of computation requirement T at a minimum throughput rate of $a_{\min}$ with an initial system configuration (x, y). Therefore

$$E_{ff}(s) = \sum_{i,j} \quad R(t, T \mid a \geqslant a_{\min}),$$

Table 1. Pr $\{i, j; t\}$ for various normalised values of $\lambda_p t$.

$\lambda_p t$	y	x				
		5	4	3	2	1
0·2	5	0·082084	0·090869	0·040237	0·008908	0·000986
	4	0·143590	0·158956	0·070386	0·015583	0·001725
	3	0·100472	0·111225	0·049251	0·010904	0·001207
	2	0·035151	0·038913	0·017230	0·003814	0·000422
	1	0·006149	0·006807	0·003014	0·000667	0·000074
0·4	5	0·006737	0·016569	0·016298	0·008010	0·001971
	4	0·027696	0·068110	0·066996	0·032950	0·008102
	3	0·045540	0·111989	0·110158	0·054178	0·013323
	2	0·037439	0·092068	0·090563	0·044541	0.010953
	1	0·015389	0·037845	0·037226	0·018309	0·004502
0·6	5	0·000553	0·002273	0·003738	0·003073	0·001263
	4	0·004036	0·016592	0·027281	0·022428	0·009219
	3	0·011783	0·048435	0·079639	0·065473	0·026913
	2	0·017198	0·070696	0·116242	0·095565	0·039282
	1	0·012551	0·051594	0·084833	0·069743	0·028668
0·8	5	0·000045	0·000278	0·000681	0·000835	0·000511
	4	0·000526	0·003227	0·007910	0·009694	0·005940
	3	0·002443	0·014975	0·036705	0·044984	0·027564
	2	0·005670	0·034744	0·085161	0·104368	0·063953
	1	0·006577	0·040305	0·098792	0·121073	0·074190
1·0	5	0·000004	0·000032	0·000110	0·000189	0·000162
	4	0·000065	0·000557	0·001915	0·003291	0·002827
	3	0·000452	0·003881	0·013337	0·022918	0·019690
	2	0·001573	0·013513	0·046438	0·079794	0·068554
	1	0·002738	0·023524	0·080842	0·138909	0·119343

Table 2a. Normalised computing power—Instruction execution rate ($P_e(i,j)$)

y	x				
	5	4	3	2	1
5	3·361600	2·952000	2·440000	1·800000	1·000000
	1·000000	0·878153	0·725845	0·535459	0·297477
4	3·050780	2·734300	2·312500	1·750000	1·000000
	0·907538	0·813415	0·687916	0·520585	0·297477
3	2·604930	2·407400	2·111000	1·666000	1·000000
	0·774900	0·716147	0·628000	0·495590	0·297477
2	1·937500	1·875000	1·750000	1·500000	1·000000
	0·576362	0·557770	0·520585	0·446216	0·297477
1	1·000000	1·000000	1·000000	1·000000	1·000000
	0·297477	0·297477	0·297477	0·297477	0·297477

Table 2b. System effectiveness for various values of $a_{\min}$

$\lambda_p t$	$a_{\min}$						
	0·9	0·8	0·7	0·6	0·5	0·4	0·2
0·2	0·225674	0·475499	0·727433	0·847070	0·962855	0·977573	0·998623
0·4	0·034433	0·119112	0·292939	0·470093	0·731123	0·829842	0·977462
0·6	0·004589	0·023454	0·087410	0·194330	0·423967	0·585005	0·909071
0·8	0·000571	0·004076	0·022175	0·066790	0·202893	0·352245	0·791149
1·0	0·000067	0·000656	0·005098	0·020350	0·085352	0·188064	0·644628

$$= \sum_{i,j} P_C(i,j)\, P_r\{i,j;t\} \text{ for } t \leqslant T,$$

$$a \geqslant a_{\min}$$

$$= \sum_{i,j} P_C(i,j \mid a \geq a_{\min})\, P_r\{i,j;t\}. \tag{2}$$

A multiprocessor system with 5 processors and 5 memory modules is considered. The probability of being in any node (i, j) i.e. $P_r\{i, j; t\}$ at any time is calculated and given in table 1 for various normalised values of λ_p and λ_m, the typical values being $\lambda_p = 20 \times 10^{-6}$ and $\lambda_m = 30 \times 10^{-6}$. The normalised computing power at various nodes (i, j) are computed and given in table 2a. The system effectiveness of the above computing system taking into account the interaction of processor and memory modules for various values of $a_{\min}$ is calculated according to equation (2) and given in figure 20 and table 2b.

8. Optimisation

The optimisation of an avionics computer system should ensure adequate performance together with an assured reliability level under constraints such as space, weight, power and cost (Lundh 1974). The peculiar problem in this particular application is that the designer may not be able to arrive at the performance requirements unless most of the design is done. The optimisation is achieved by partitioning the system into small parts which may have a great deal of inter-dependence among them. These smaller subsystems are designed and optimised simultaneously.

When a multiprocessor system is used for aerospace applications, the program memory is partially replicated to reduce the queuing delays caused by memory contention. The real time software for aerospace application is normally divided into smaller sections, some of which may have to be referred to very often and some very rarely. To reduce the queuing delays associated with memory contention, the sections which are referred to very often may have to be replicated more than the rest of the program sections. The extent of replication required under the performance constraint becomes an optimisation problem. Covo (1974) optimises the number of processor and memory modules required with constraints on both the processor utilisation and the cost. Processor utilisation is related to memory contention, which is modelled by combinatorial techniques. Also the software modules are arranged according to the speed size characteristics. Covo considers an organisation in which the program memory is a common pool consisting of size X_j and is replicated n_j times $(n_j \leqslant m), j = 1, .., k$. Kachal & Arora (1975) minimise the product function of the total cost and elapsed time for a multiprogrammed system with several levels of memory hierarchy. Actually the number of program modules in each level is optimised. Here again the I/O activity of the program modules plays an important role (i.e.), the program modules which are going to be referenced very often are stored in the lower hierarchical level. In this analysis, a queuing model is used to find the probability of the CPU being busy. The pattern search technique is used for optimisation. The optimisation problem is solved in two steps.

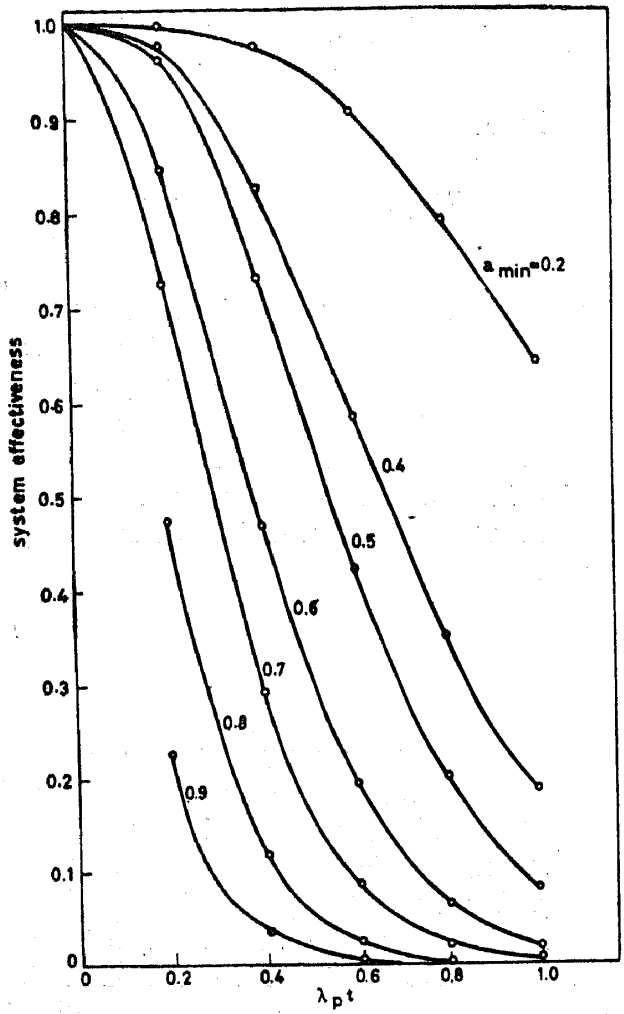

Figure 20. Multiprocessor system effectiveness for various values of a_{min}

The first step determines the best configuration through unconstrained optimisation. In the second step, the best unconstrained design is altered to include cost and response time constraints.

In the above two optimisation models, it is assumed that the various modules of the computer system operate perfectly and hence the delay in throughput is attributed to memory contention due to program behaviour. However, in practical systems like airborne computing systems and electronic exchanges, the modules of the computing system can fail. As the modules may fail during operation, the memory contention is very much affected. The authors modify the optimisation model of Covo to take care of the changes in memory contention due to module failures (Pedar 1980). The system diagram is given in figure 21. Only processor and memory modules are taken for consideration assuming that the interconnection link is perfect. The graph model for the computing system which takes into account the possible failures of processor and memory modules is given in figure 22. This diagram corresponds to the j th stage of the memory having n_j memory modules and m processor modules.

The probability of the system being at any of the above state denoted by $P_r\ (i, j, t)$ is given by the combinatorial reliability equation (1). The expected value of ψ_j taking reliability into account is given by

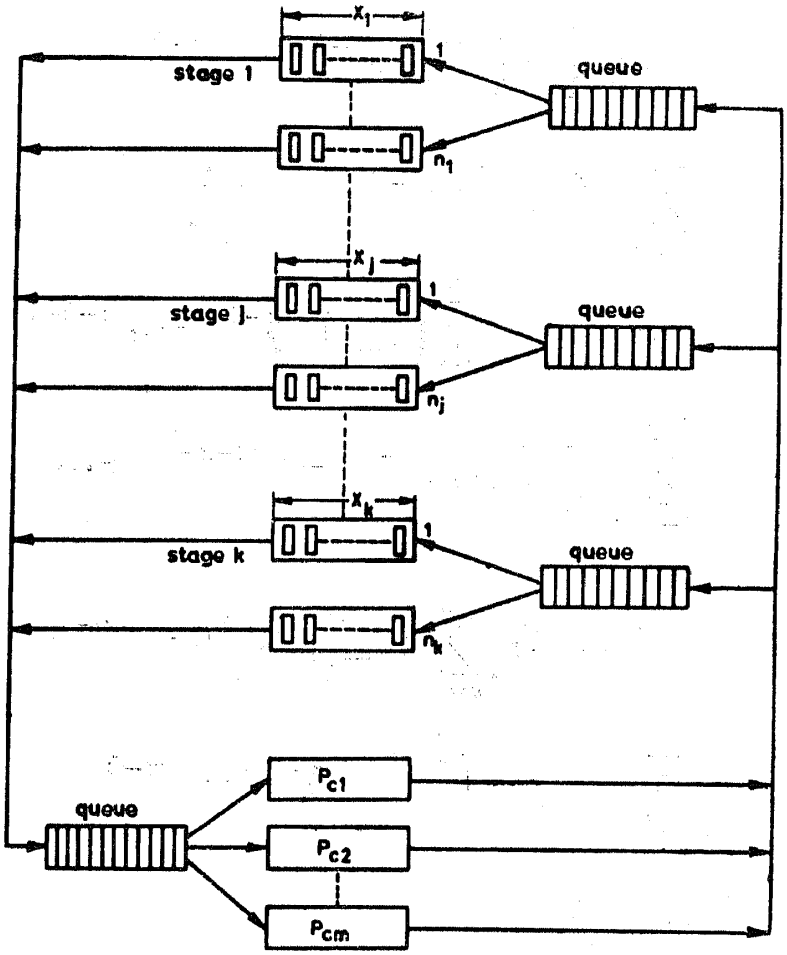

Figure 21. Queueing model for multiprocessor system of figure 2

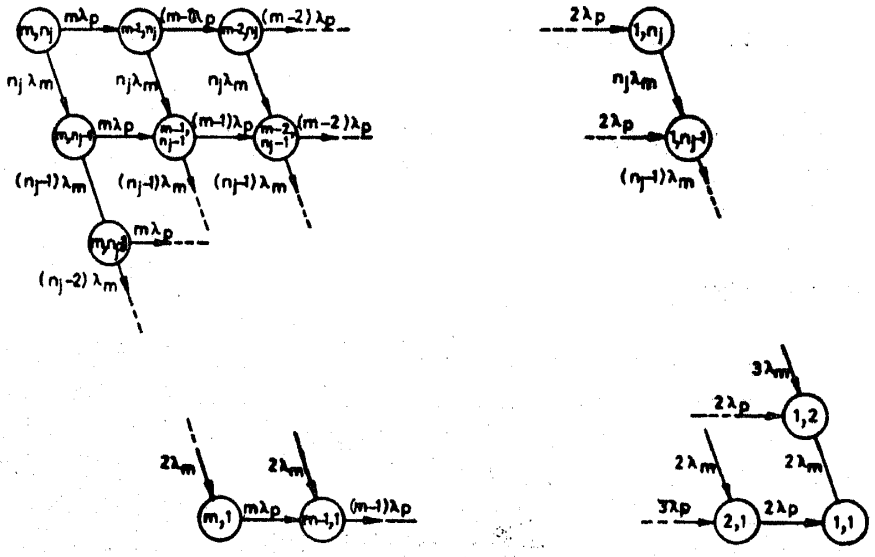

Figure 22. Multiprocessor system states

$$\psi_j = P_r\{m, n_j\}\left[\sum_{k=1}^{n_j}\binom{m-1}{k-1} p_j^{k-1}(1-p_j)^{m-k} + \sum_{k=n_j+1}^{m}\frac{k}{n_j}\binom{m-1}{k-1} p_j^{k-1}(1-p_j)^{m-k}\right]$$

$$+ P_r\{m-1, n_j\}\left[\sum_{k-1}^{n_j}\binom{m-2}{k-1} p_j^{k-1}(1-p_j)^{m-k-1} + \sum_{k=1_j+1}^{m-1}\frac{k}{n_j}\binom{m-2}{k-1} p_j^{k-1}(1-p_j)^{m-k-1}\right]+\cdots$$

$$\vdots$$

$$+ P_r\{m, n_j-1\}\left[\sum_{k=1}^{n_j-1}\binom{m-1}{k-1} p_j^{k-1}(1-p_j)^{m-k} + \sum_{k=(n_j-1)+1}^{m}\frac{k}{n_j-1}\binom{m-1}{k-1} p_j^{k-1}(1-p_j)^{m-k}\right]$$

$$\vdots$$

$$+ P_r\{1, 1\}[1].$$

Hence the optimisation problem becomes

$$\text{Minimise} \quad Z = A_0 m + \sum_{j=1}^{K}(C_1 + C_2 X_j)n_j$$

where Z is the cost function subject to

$$\sum_{j=1}^{K} p_j \psi_j \leq \rho_{\max};\ mp_j \leq n_j \leq m;\ j=1, 2,\ldots, K.$$

9. Fault tolerant software

The hardware fault tolerant architectures so far described are aimed at reducing the effect of component failures rather than design faults. On the contrary, as there is no aging in software, the software fault tolerance is aimed at reducing the hazardous effect of the design faults (Wulf 1975). These software design faults have become increasingly prevalent because of the steadily increasing size and complexity of the software systems. In this section, we briefly review the various software reliability models and a few suggested software fault tolerance techniques.

9.1 *Software reliability models*

Software faults are defined as 'deviations from correct program execution which are due to errors occurring during the translation of the original specification of an algorithm to the program being executed' (Anderson *et al* 1979). Software reliability models are simple mathematical abstractions of software faults and consequent failures and are used to assess the reliability of software systems systematically.

9.1a *Shooman's model* Assuming that there are E_T errors in the program and during a debugging time τ, $\mathscr{E}_c(\tau)$ errors are found and debugged, then the remaining number of errors is given by Shooman (1973) as

$$\mathscr{E}_r(\tau) = \frac{E_T}{I_T} - \mathscr{E}_c(\tau)$$

where I_T is the total number of machine language instruction. It is also assumed that the hazard function $Z(t)$ is proportional to the number of errors remaining

$Z(t) = C\,\mathscr{E}_r(\tau)$, which leads to

$$R(t, \tau) = \exp\ [-C(E_T/I_T) - \mathscr{E}_c(\tau)t].$$

This reliability can be evaluated with the estimates of $\hat{C}$ and $\hat{E}_T$ through a number of debugging trials.

9.1b *Jelinski-Moranda model* This model assumes that the debugging time between errors is exponentially distributed (Jelinski & Moranda 1973) giving rise to the density function for the time of discovery of the ith error as

$$P(t_i) = \lambda_i \exp\ [(-\lambda_i\, t_i)],$$

where $\lambda_i = \Phi\,(N - i + 1)$ and $N =$ number of errors present initially.

9.1c *S-W model* The Jelinsky-Moranda model is slightly modified by Schick & Wolverton (1973) to take care of the debugging time interval. This gives rise to a hazard function of type

$$Z(t_i) = \Phi\ [N - (i-1)]\ t_i,$$

where $t_i =$ time between ith and $(i-1)$st errors discovered.

9.1d *Nelson model* This is a reliability model based on data domain (i.e.) based on error correction with program structure. The function of the program is to evaluate a value of $f(E_i)$ where $E = \{E_i;\ i = 1, 2, \ldots, n\}$ is the input data set. Suppose there is some program error. Then for all inputs in a subset $\overline{E}_e$ of E, the program will yield incorrect outputs and for other inputs sets E_e, it yields a correct

output. If n_e is the number of E_i in E_e, then n_e/N may be interpreted as the probability that a run of a program with an input data set selected at random from E will give the correct output (Schick & Wolverton 1978). Then the reliability model is given by

$$R=1-(n_e/N).$$

There are some more variations of the above models like the ones due to Musa (1975), Corcoran *et al* (1954), Weiss (1956) and Basin (1973). These models are described in detail in Schick & Wolverton (1978) and Thayer *et al* (1978).

9.2 *Software fault tolerance*

It is a highly difficult job to have a software without any design faults, which may be attributed to one or more of the following reasons (Hetch 1976): (i) existing validation methods do not indicate the completion of entire debugging (ii) no formal methods to guarantee correct operation because of the possible unmanageable number of states of software systems, (iii) the best of the current methods are extremely costly (iv) bad data and/or use environment may cause anomalies in the execution and (v) there is no generally accepted methodology for limited verification after the change of a program. However, it is possible to use the error seeding methods of Gilb (1977) and Krishnamurthy & Srinivasan (1979) to find an estimate of the total number of errors remaining in the software to gain a confidence. Now that the program is always prone to some form of design faults, it always becomes mandatory for the real time critical system designer to incorporate fault-tolerance in software systems because the design faults cannot be covered by the hardware fault tolerance techniques even though some hardware faults can be handled by software to a limited extent.

The hardware fault tolerance is mainly based on the static structure analysis of the faults. In the same way, the dynamic structure analysis, based on algorithmic flow, which is based on activity analysis which in turn is based on the atomic action gives an insight into the effects of the faults and the ways of tolerating them (Randell *et al* 1978).

Fault tolerance in operating systems are incorporated at language design level. Some intentional blocks will be included, 'signals' and 'exception handlers' are used to cope with anticipated but unusual situations and 'acceptance tests' and 'recovery blocks' are used for unanticipated situations. Signals and exception handlers are however useful only in on-line systems where the user is indicated of errors and possible corrective action (Anderson *et al* 1979). For a real time situation, the best possible approach is the acceptance tests and recovery blocks of Randell (1975). Here the large program is segmented into functional blocks. Each functional block has an acceptance test and a sequence of alternate blocks. The error in the block is indicated by the acceptance tests and the alternate block is switched in, which may or may not be the same copy but containing the same function. There are several studies using the recovery block scheme and a recent study indicates the optimal way of using this scheme for operating systems and a method of computing coverage (Gannon & Shapiro 1978). On the real time application program side, a scheme similar to that of Randell is approached by Hetch (1976) with particular

application to aircraft navigational systems. Reliability Markov model is given for this scheme used in such applications. Fault-tolerant software techniques are very much in an infant stage and one could expect a lot of advances in this area in the near future.

10. Concluding remarks

The advances in semiconductor technology have made a digital computing system one of the essential equipment for society. The ultimate requirements in digital computing system technology is the reliability and safety of the systems. The present hardware technology is equipped with all the capabilities to build the required fault tolerance into the digital computing systems. On the other hand a unified and standard method for the design, analysis and validation of the fault-tolerance is still young, largely unexplored and undeveloped discipline and it is essential to develop this area to convince and educate the potential users. Also present-day hardware technology has advanced to such a stage that software has become the only costly item of a computing system. Fault-tolerant software techniques are at an infant stage.

Current research directions in this area are towards unified reliability models, optimal resource allocation, new measures such as performability for degradable computing systems, phased mission approach, multistate models, self-checking circuits, time-dependent and state-dependent coverage models and the applications of fault tree models for coverage. This survey paper has attempted to highlight some of the current attempts in the areas of design, analysis and validation of FTC systems.

The authors thank Dr S R Valluri, Mr C S Rangan and Prof. A K Rao, for their encouragement during the course of this work. This work is partially supported by an ARDB grant-in-aid project 'Airworthiness of aircraft—a system engineering approach.'

List of symbols

A	transition probability matrix
a	computing power
c	coverage
C	computer subset
c_1, c_2	cost
$\mathscr{C}$	computer class
d_j	fraction of total faults belonging to class j
F	set of faults of computer C
H	state of the computer with faults
K	number of stages in each memory module
N	number of computers available initially

$P_i(t)$	probability of system being at state i at time t
$\dot{P}_i(t)$	$\frac{d}{dt}(P_i(t))$
$\bar{P}(t)$	vector of $P_i(t)$
$\bar{P}^T(t)$	transpose of $\bar{P}(t)$
$P_r\{i, j; t\}$	$P_r\{$Multiprocessor system being at state (i, j) at time $t\}$
$P_r\{g, h; t\}$	$P_r\{$Multiprocessor system being at state (g, h) for task length $T\}$
$P_C(i, j)$	computation power at state (i, j)
$p(t_i)$	density function for the discovery of ith error
$R(t)$	reliability
$R^*(N, S, Ac)[t]$	reliability for N active modules, S spare modules and algorithmic coverage Ac
$R(t, T/a \geqslant a_{\min})$	performability
$R(t, \tau)$	software reliability
$R_i^c(t)$	computation-based reliability
$R\sigma(C)$	reliability with σ tolerance
$R_m(t)$	memory module reliability
$R_p(t)$	processor module reliability
S	number of spares
t, T	time variables
x	number of processor modules available initially
y	number of memory modules available initially
β	transition function
Δ	transition function
$\{\delta_0, \delta_1 \ldots\}$	sequence function
σ_{ij}	transition probability from state i to j
σ_i	unconditional transition probability out of state i
λ	failure rate
μ	repair rate
ν, τ	time variables

References

Anderson T, Lee P A & Shrivastava S K 1979 *System fault tolerance—computing systems reliability* eds T Anderson and B Randell (Cambridge: University Press)

Avizienis A 1977 *Proceedings informatian processing* (Amsterdam: North Holland) p. 405

Avizienis A 1978 *Proc. IEEE* **66** 1109

Basin S L 1973 Software reliability report, Science Applications Inc., Palo Alto, California

Beaudry M D 1977 *Proc. 7th Int. Conf. in Fault Tolerant Computing* (New York: IEEE Press) p. 75

Bellon C & Saucier G 1976 *Proc. 2nd EUROMICRO Symposium on Microprocessing and Microprogramming* (Amsterdam: North Holland) p. 76

Bennetts R G 1978 *Electron. Power* **24** 51

Bhandarker D P 1975 *IEEE Trans. Comput.* **C24** 897

Bjurman B E, Jenkins G M & Mesreliez C J 1976 NASA Report No. CR-145024, NTIS, Washington

Borgersen B R & Freitas R E 1975 *IEEE Trans Comput.* **C24** 517

Bouricius W G, Carter W C & Schneider P R 1969 *Proc. Natl. Conf. of Association for Computing Machinery*, p. 295

Bricker J L 1973 *IEEE Trans. Reliab.* **R22** 72

Brouaeys F & Muron O 1977 *Proc. III Int. Symp. on Measuring, Modelling and Evaluating Computer Systems*, (Amsterdam: North Holland) p. 375
Conn R B, Merryman P M & Whitelaw K L 1977 AGARD-O graph 224 (NTIS, Washington) p. 6.1
Corcoran W J, Weingarten H & Zehra P W 1954 *Manage. Sci.* **10** 786
Courtois B 1976 *Proc. VI Annu. Int. Conf. on Fault-Tolerant Computing* (New York: IEEE Press) p. 53
Covo A A 1974 *IEEE Trans. Comput.* **C23** 113
Dorrough D C 1971 *IEEE Trans. Reliab.* **R20** 169
Gannon T F & Shapiro S D 1978 *IEEE Trans. Software Engg.* **SE4** 390
Gilb T 1977 *Software metrics* (Cambridge, Mass: Winthrup) p. 28
Goldberg J 1974 *Proc. Conf. on Reliability and Fault Tree Analysis* (Philadelphia: Soc. for Ind. and Appl. Maths), p. 687
Hetch H 1976 *ACM Computing Surveys* **8** 391
Holick A 1963 *IEEE Trans. Electron. Comput.* **12** 365
Hoogendoorn C H 1977 *IEEE Trans. Comput.* **C26** 998
Hopkins A L 1977 *Proc. Joint Automatic Control Conf.* (New York: IEEE Press) p. 232
Hopkins A L, Smith B and Lala J H 1978 *Proc. IEEE* **66** 1221
Ingle A D & Siewiorek D P 1977 *Proc. VII Annu. Int. Conf. on Fault-Tolerant Computing* (New York: IEEE Press) p. 3
Jacques Losq 1974 Tech. Note No. 33, Digital Systems Laboratory, Stanford University
Jacques Losq 1976 *IEEE Trans. Comput.* **C25** 569
Jacques Losq 1977 *Proc. VII Annu. Int. Conf. on Fault-Tolerant Computing* (New York: IEEE Press) p. 29
Jelinski Z & Moranda P B 1973 *Proc. IEEE Symp. on Computer Software Reliability*, (New York: IEEE Press) p. 78
Kaachal S K & Arora S R 1975 *Proc. Annu. Conf. ACM*, p. 96
Krishnamoorthy E V & Srinivasan G L 1979 *Proc. I Int. Conf. on Reliability and Exploitation of Computer Systems*, Wroclaw, Poland
Koczela L J & Burnet G J 1968 *IEEE Trans. Aerosp. Electron. Syst.* **AES4** 456
Kuehn R E 1969 *IEEE Trans. Reliab.* **R18** 3
Landrault C & Laprie J C 1977 *Proc. VII Annu. Int. Symp. on Fault-Tolerant Computing*, p. 10
Lundh Y 1974 *AGARD-ograph*-183 p. 42
Mathur F P 1971a *JPL Q. Tech. Rev.* Vol. 1
Mathur F P 1971b *IEEE Trans. Comput.* **C20** 1376
Meyer J F 1976 *IEEE Trans. Comput.* **C25** 578
Meyer J F 1978 *Proc. VIII Annu. Int. Conf. Fault-Tolerant Computing*
Moore E F & Shannon C E 1956 *J. Franklin Inst.* **262** 191
Murray N D, Hopkins A L & Wensley J H 1977 AGARD-ograph 224 p. 171
Musa J D 1975 *IEEE Trans. Software Engg.* **SE1** 312
Ng Y W 1976 *Reliability modelling and analysis for fault-tolerant computers*, Ph.D. Thesis, Univ. of California, Los Angeles
O'Hern E A 1975 *Acta Astronaut.* **2** 667
Pedar A & Sarma V V S 1978 *IEEE Trans. Reliab.* **R27** 345
Pedar A 1980 Ph.D. thesis, Indian Institute of Science (in preparation)
Perry J L, Stansberry J R & Dennis N G 1972 Tech. Phase Rep. No. 2, SCI Systems Inc., Huntsville, Alabama, USA
Ramamoorthy C V & Han Y W 1975 *IEEE Trans. Comput.* **C22** 868
Ramamoorthy C V, Han Y W & Ho G S 1977 *Proc. Natl. Comput. Conf.*, (Montvale, NJ: AFIPS Press) p. 13
Rennels D A 1978 *Proc. IEEE* **66** 1255
Rennels D A & Avizienis A 1973 *Proceedings III Annu. Int. Symp. on Fault-Tolerant Computing*, p. 131
Randell B 1975 *IEEE Trans. Software Engg.* **SE1** 220
Randell B, Lee P A & Treleaven P C 1978 *ACM Computing Surveys* **10** 123
Schick G J & Wolverton R W 1973 *Proceedings operations research* (Würzburg: Physica Verlag) p. 395
Schick G J & Wolverton R W 1978 *IEEE Trans. Software Engg.* **SE4** 104

Shooman M L 1973 *Proc. IEEE Symp. on Computer Software Reliability* (New York: IEEE Press) p. 51
Short R A 1968 *IEEE Comput. Group News* **2** 2
Sousa P 1976 *Modular redundancy techniques for fault-tolerant digital systems*, Ph.D. thesis, University of Missouri, Columbia
Stiffler J J 1975 Raytheon Rep. ER-75-4293, Raytheon Company, USA
Strecker W D 1970 *Analysis of the instruction execution rate in certain computer structures*, Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, USA
Su S Y H & Ducasse E 1975 *Proc. Int. Comput. Symp.* (Amsterdam: North Holland) p. 216
Su S Y H & Spillman R J 1977 *Proc. Natl. Comput. Conf.* (Montvale, NJ: AFIPS Press) p. 19
Teoste R 1964 *IEEE Trans. Reliab.* **R13** 42
Thayer T A, Lipow M & Nelson E C 1978 *Software reliability* (Amsterdam: North Holland)
Trivedi A K & Shooman M L 1974 Research Rep. No. EE/EP 74-011-EER-110, Polytechnic Inst. Brooklyn
Troy R 1977 *Proc. VII Annu. Int. Conf. on Fault-Tolerant Computing*, p. 44
von Neumann J 1956 *Ann. Math. Stud.* **34** 43
Weiss 1954 *Oper. Res.* **4** 532
Wulf W B 1975 *IEEE Trans. Software Engg.* **SE2** 233